Automated Data Linkage

Nov 28th, 2024

Title Automated Data Linkage – Developer Facing Documentation

Version 0.2.22

Description The automated data linkage documentation is meant to provide an in-depth overview of the files/scripts, functions, and logic to ensure the maintainability and potential future maintenance of the package is easier to understand and follow. A breakdown of the architecture, libraries, script logic, and script functionality will be provided

Depends R (>= 4.4.1)

Built On R 4.4.1

Encoding UTF-8

Language en-CA

Author Cole Chuchmach [aut/cre]
Barret Monchka [aut/ctb]

Maintainer The George & Fay Yee Centre for Healthcare Innovation

Date/Publication 2024-11-28 12:00:27 CST

R topics documented

populate_linkage_metadata.R

Description

data_linkage_scripts.R is the script that will create an **SQLite** file for storing metadata related to your datasets and linkage algorithms.

Details

The **SQLite** file generated is used in both the **Linkage Metadata GUI**, and the programmatic **run_main_linkage()** function. The generated file will initially be blank, with some default values already pre-populated for selection.

The first step of the script involves creating a blank SQLite file, and running **CREATE TABLE** queries to establish table dependencies for the metadata UI and main linkage script.

Once all the table dependencies are set up, default data is pre-populated into the SQLite file through additional queries, this includes linkage methods, acceptance methods, comparison methods, and linkage rules.

linkage_helper_functions.R

Description

data_linkage_scripts.R is the script that provides the user with many functions called before, after, or during data linkage to acquire specific database information, apply data standardization, or create pre-processing options.

Details

The available specifications are:

- get_blocking_keys()
- get_matching_keys()
- get_acceptance_thresholds()
- get_linkage_technique()
- get_linkage_technique_description()
- get_implementation_name()
- get_iteration_name()
- get_algorithm_name()
- get_ground_truth_fields()
- get_linkage_output_fields()
- apply_output_cutoffs()
- get_standardized_names()
- load_linkage_file()
- create_extra_parameters_list()

get_blocking_keys()

Description

get_blocking_keys() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a data frame of the blocking keys and linkage rules.

Usage

```
get_blocking_keys(linkage_metadata_conn, 2)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to limit the blocking keys for that specific iteration.

Once the query is returned, additional cleaning is performed to replace linkage rule IDs with key value pairs stored in a JSON format.

A data frame is returned containing **three** columns, the field name of the blocking pair in the left dataset, the field name of the blocking pair in the right dataset, and key-value pairs of what linkage rules should be applied during data linkage. If no blocking keys exist for the iteration, an empty data frame is returned.

---

get_matching_keys()

Description

get_matching_keys() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a data frame of the matching keys, linkage rules, and comparison rules.

Usage

```
get_matching_keys(linkage_metadata_conn, 2)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to limit the matching keys for that specific iteration.

Once the query is returned, additional cleaning is performed to replace linkage rule IDs and comparison rule IDs with key value pairs stored in a JSON format.

A data frame is returned containing **four** columns, the field name of the matching pair in the left dataset, the field name of the matching pair in the right dataset, and key-value pairs of what linkage and comparison rules should be applied during data linkage. If no matching keys exist for the iteration, an empty data frame is returned.

get_acceptance_thresholds()

Description

get_acceptance_thresholds() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a list of key value pair acceptance thresholds.

Usage

```
get_acceptance_thresholds(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to obtain all the parameter keys and values.

Once the query is returned, additional cleaning is performed to create the key-value pairs, appending each to a list which will be returned.

A list is returned, where each acceptance value is a key-value pair of the variable name and its numeric value. If no acceptance threshold was found under the provided iteration ID, an empty list (*list of length 0*) is returned.

get_linkage_technique()

Description

get_linkage_technique() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a character string of the technique.

Usage

```
get_linkage_technique(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to obtain the technique label being used within the class being called for this iteration.

A character string of the linkage technique is returned if there exists an iteration under this ID, otherwise, NA is returned.

---

get_linkage_technique_description()

Description

get_linkage_technique_description() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a character string of the technique's description.

Usage

```
get_linkage_technique_description(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to obtain the description of the technique label being used within the class being called for this iteration.

A character string of the linkage technique description is returned if there exists an iteration under this ID, otherwise, NA is returned.

---

get_implementation_name()

Description

get_implementation_name() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a character string of linkage implementation which will be called when running that iteration of an algorithm for data linkage.

Usage

```
get_implementation_name(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to obtain the implementation name to identify which class to call and pass parameters to for data linkage.

A character string of the implementation name is returned if there exists an iteration under this ID, otherwise, NA is returned.

---

get_iteration_name()

Description

get_iteration_name() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific pass of an algorithm. Returning a character string of the identifiable iteration name defined by the user.

Usage

```
get_iteration_name(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.

Details

The function performs a query on the provided database connection, using the iteration ID to obtain the identifying iteration name for summary or timing purposes.

A character string of the iteration name is returned if there exists an iteration under this ID, otherwise, NA is returned.

---

get_algorithm_name()

Description

get_algorithm_name() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific algorithm. Returning a character string of the identifiable algorithm name defined by the user.

Usage

```
get_algorithm_name(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
algorithm_id	An integer ID specific to an algorithm.

Details

The function performs a query on the provided database connection, using the algorithm ID to obtain the identifying algorithm name for summary or timing purposes.

A character string of the algorithm name is returned if there exists an algorithm under this ID, otherwise, NA is returned.

---

get_ground_truth_fields()

Description

get_ground_truth_fields() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific algorithm. Returning a data frame of the left and right dataset fields, along with any linkage rules they may be using.

Usage

```
get_ground_truth_fields(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
algorithm_id	An integer ID specific to an algorithm.

Details

The function performs a query on the provided database connection, using the algorithm ID to obtain the ground truth fields of the algorithm.

Additional work is performed on the data frame before returning it such as replacing the comparison IDs with their key-value pairs.

A data frame of the left and right dataset fields is returned along with any comparison rules if there exists an algorithm under this ID, otherwise, an empty data frame is returned.

get_linkage_output_fields()

Description

get_linkage_output_fields() will take in a connection to the linkage database you plan on using, along with an integer which correlates to a specific algorithm. Returning a data frame of the output labels and their associated field names (*from the left dataset*).

Usage

```
get_linkage_output_fields(linkage_metadata_conn, 1)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
algorithm_id	An integer ID specific to an algorithm.

Details

The function performs a query on the provided database connection, using the algorithm ID to obtain the output fields of the algorithm.

A data frame of the label and field name pairs is returned if there exists an algorithm under this ID, otherwise, an empty data frame is returned.

apply_output_cutoffs()

Description

apply_output_cutoffs() will take in a connection to the linkage database you plan on using, an integer which correlates to a specific algorithm, and a data frame of values to apply cutoffs to. Returning an updated version of the data frame passed with cutoffs apply to certain values.

Usage

```
apply_output_cutoffs(linkage_metadata_conn, 1, input_data)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
algorithm_id	An integer ID specific to an algorithm.
output_df	A data frame of the output fields and associated values obtained during data linkage.

Details

The function performs a query on the provided database connection, using the algorithm ID to identify how to modify the input data frame and apply cutoffs to group many values together for easier comparisons.

An updated version of the input data frame is returned if there exists an algorithm under this ID, otherwise, no modifications are made. There are currently **nine field types**, their specific details are as follows:

Type 1 – Generic/Pass Through Field: The field is passed through as is, with no cutoffs applied to the field.

Type 2 – Date (Year): Meant for a field in a data frame comprised of year values (*e.g., birth year, acquisition year, etc*.), years are grouped into cutoffs of 8 years, with the lowest being **<1975** and greatest being **>2044**.

Type 3 – Age: Meant for a field in a data frame comprised of age values. Ages are grouped into the ranges of **<18, 18-34, 35-64, 65-80,** and **81+**.

Type 4 – Postal Code Initial: Will extract the initial of a postal code and group them by the first digit.

Type 5 – Name Length: Will take the character count of the field and group by the number of characters in a person's name, with the lowest cutoff being **<5**, and greatest being **8+**.

Type 6 – Name Count: Will take the number of names in a single cell of the the field and group by how many different names a person has, with the lowest cutoff being **1**, and the greatest being **3+**.

Type 7 – Derived Age: Will combine two fields into one, taking the different of a **capture date** and **birth date** field, apply the same age cutoffs as **type 3**.

Type 8 – Standardization File: The field pass will contain some sort of value that can be replaced using a file provided through the GUI. Each of the values in the data frame will be replaced with their corresponding value in the lookup file provided. If no mapping is found for a value, the original value will be used in place of a standardized value.

Type 9 – Forward Sortation Area (FSA): Will take the first character of a postal code and replace it with their respective provinces. If a character has no match, it is marked as **Missing**.

get_standardized_names()

Description

get_standardized_names() will take in a connection to the linkage database you plan on using, an integer which correlates to a specific iteration of an algorithm, and a vector of names to standardize. Returning a vector of unique names standardized to a common spelling.

Usage

```
get_standardized_names(linkage_metadata_conn, 1, data_field)
```

Arguments

linkage_db	A database connection to the linkage metadata file being used for data linkage.
iteration_id	An integer ID specific to an algorithm's iteration.
data_field	A vector of unique names to standardize.

Details

The function performs a query on the provided database connection, using the iteration ID to identify how to modify the input vector of names, using a user defined look up table, or the packages default lookup values if none is found.

An updated version of the person's name is returned if there exists a value in the lookup table that matches the input name, otherwise, NA is returned in place of the name.

load_linkage_file()

Description

load_linkage_file() will take in a file path of types **csv**, **txt**, **sas7bdat**, **xlsx**, **sqlite** (*datastan output*), and will return a data frame of the read in data.

Usage

```
load_linkage_file(file_path)
```

Arguments

dataset_file	A path to the dataset file that should be loaded in and returned as a data frame.

Details

The function checks for the file extension of the passed file path, and uses that extension to determine which package will read in the data and return it as a data frame.

create_extra_parameters_list()

Description

create_extra_parameters_list() will allow users to modify any of the default parameter values to allow for more specific linkage options such as exporting or saving certain linkage results.

Usage

```
load_linkage_file(...)
```

Arguments

linkage_output_folder	An output directory where results will be stored during the linkage process.
include_unlinked_records	Includes the unlinked records in the final linkage results data frame. **(Options: TRUE, FALSE)**
output_linkage_iterations	Exports the linked iteration pairs for each pass of an algorithm, which includes the pair values and their scores. **(Options: TRUE, FALSE)**
output_unlinked_iteration_pairs	Exports the unlinked iteration pairs for each pass of an algorithm, including their scores. **(Options: TRUE, FALSE)**
linkage_report_type	Determines what type of report should be generated after an algorithm has been run. **(Options: 1 = No Report/CSV Linkage Results, 2 = Intermediate Report for All Algorithms Being Ran, 3 = Final Report for Each Algorithm Being Ran)**
calculate_performance_measures	Exports the performance measures of each algorithm being ran, which includes PPV, NPV, Sensitivity, Specificity, F1-Score, and Linkage Rates. **(Options: TRUE, FALSE)**
data_linker	The name of the person performing the linkage.
generate_algorithm_summary	Exports the algorithm summary of each algorithm being ran, which includes each passes blocking and matching criteria, acceptance threshold, technique, and linkage rate. **(Options: TRUE, FALSE)**

generate_threshold_plots	Exports PNG images of plots taken to help establish valid acceptance thresholds, or to show an acceptance threshold cutoff once selected. **(Options: TRUE, FALSE)**
save_all_linkage_results	Saves the linkage results, algorithm summary, performance measures, and missing data indicators in an **.Rdata** file in the packages **AppData** which can be used to regenerate reports for later use. **(Options: TRUE, FALSE)**
collect_missing_data_indicators	Will browse the input dataset for missing values in the output fields included in the linkage results and keep them for report purposes. **(Options: TRUE, FALSE)**
save_audit_performance	Will perform a query after an algorithm finishes completion which involves writing its performance measures to the database so that it may be viewed or exported for later use. **(Options: TRUE, FALSE)**

Details

The function will accept any number of input values and will override the default NULL or FALSE values with user input, returning a list of the parameters which can be passed during linkage.

data_linkage_scripts.R

Description

data_linkage_scripts.R is the script that will perform the automated data linkage by dynamically calling linkage classes which inherit an abstract class. The output and results depend on what the user would like to be exported, with more information on what they user can decide found in the **extra_parameters** variable.

The script contains **one** main function with **2** default linkage classes to perform the linkage process.

Details

The data linkage script is paired with an **SQLite** metadata file to pull linkage information, including algorithms, passes, the fields it uses, and what output to keep. Of which requires valid datasets and algorithms to be enabled in the database.

To perform data linkage using metadata information, call the **run_main_linkage()** function and supply **five** input parameters.

- **Left Dataset:** The left dataset should be a file of type csv, excel, txt, SQLite, etc. Where the prefix of the file name should match the dataset code entry in the database.
- **Right Dataset:** The right dataset should be a file of type csv, excel, txt, SQLite, etc. Where the prefix of the file name should match the dataset code entry in the database.
- **Linkage Metadata File:** The SQLite file that contains all the users linkage information (*algorithms, passes, datasets, etc*).
- **Algorithm IDs:** A vector of integers that pertain to the unique ID of each algorithm, each algorithm ID will be used to obtain the individual passes, and output of each algorithm.
- **Extra Parameters:** A list of logical, numeric, and character values used to dictate what kind of data to export to the user, reports to generate, and values to calculate.

Additionally, the **run_main_linkage()** will return a list of relevant data if the user requested linked data to be returned within the extra parameters field, otherwise NULL is returned.

Lastly, error messages are provided to the user to help inform them on what went wrong during runtime and how they may be able to fix it.

Arguments

left_dataset_file	A path to the left dataset file, where the prefix of the file matches a dataset code in the metadata file.
right_dataset_file	A path to the right dataset file, where the prefix of the file matches a dataset code in the metadata file.
linkage_metadata_file	A path to the SQLite linkage metadata file.
algorithm_ids	A vector of integer values, each pertaining to a specific algorithm in the database.
extra_parameters	A list of key-value pairs, each of which have logical, numeric, or character value which dictates what kinds of output to export during or after data linkage takes place.

---

run_main_linkage()

Description

Run_main_linkage() is the function used to perform data linkage on the datasets and algorithms passed as parameters using the saved information in the linkage metadata file.

Details

When the function is called, the first of the main steps is to perform some error handling prior to database connection. This involves making sure that a left and right dataset, algorithm IDs, metadata file, and extra parameters were provided. Beyond the error handling, the main steps are as follows…

Step 1: Connect to the linkage metadata database to obtain access to algorithms and their individual passes. Create intermediate lists for storing the linked data, algorithms, performance, etc. of each pass so that it may be saved or returned later. **Steps 2-8** repeat for each individual algorithm.

Step 2: Obtain the iteration IDs that belong to the algorithm we're currently planning on running. This is done using a simple query for all enabled iterations.

Step 3: Load in the left and right dataset for this algorithm (*done each algorithm since we need to start fresh with each algorithm*), perform error handling by ensuring the datasets being loaded in here match fields with what's stored in the database, and by checking that output fields were provided. Lastly, if the user asked to obtain missing data indicators, the data will be checked and a data frame of logical values (*TRUE or FALSE*) is obtained for the columns the user defined to be in the results output.

Step 4: For each iteration in our list of iterations, run the iteration using the linkage implementation defined by the user, and use the results to store information, and know which value we no longer need. There are **six results** to manage.

> **Result 1:** Using the linked indices, remove these specific rows from the dataset we passed in so that we may know which rows were successfully linked on the most recently completed pass.
>
> **Result 2:** Export the data frame of pairs BEFORE the 'link' step takes place, containing the pair values, scores for observations, and whether each pair was selected/accepted.
>
> **Result 3:** Export the data frame of pairs AFTER the 'link' step takes place, containing the linked pair values, and scores for observations.
>
> **Result 4:** The output data frame (*the data frame that contains the output fields the user is interested in keeping*) is binded with the previously ran iterations.
>
> **Result 5:** Performance measures obtained during the iteration (*if a ground truth was provided and the performance measures were actually calculated*) are stored all together until we perform a final calculation.
>
> **Result 6:** Any plots generated during the iteration (*along with their captions*) are both saved and exported as PNG images that can later appear in the generated PDF reports. Note however, that the program will crash if the user moves the saved PNG images BEFORE the report is generated.

Step 5: The unlinked data is binded with the output data that was successfully linked, this dataframe contains a column of logical values that act as a link indicator for whether it was linked or not. Labels and cutoffs are then applied to the data frame using the output fields and field types defined by the user within the metadata.

Step 6: The performance measures, linked data summary, and algorithm summary are all saved and exported if requested by the user. The performance measures obtained during the linkage process are used to calculate **PPV**, **NPV**, and other measures, before exporting the data into a CSV. The linked data is either exported as a CSV if no report was requested, or as a PDF quality report using the **'linkrep'** package developed by Elizabeth Stoughton. And the algorithm summary is exported as CSV where each row contains information regarding an iteration, including the matching and blocking criteria, acceptance threshold, and linkage rate.

Step 7: The information obtained during this algorithm is saved to the list of intermediate information in preparation for an intermediate report, or if the user would like to have the linkage function return results.

Step 8: If the user wanted the performance measures to be saved for the algorithm being ran, the performance measures are stored as a JSON format within the database that can be both viewed and/or exported from within the GUI.

Final Steps: Once all algorithms have been ran, an intermediate report will be generated using all the intermediate data gathered during the function (*if the user has requested so*), all before either saving the linkage results by returning a list of all the intermediate data, or NULL if the user had chosen to opt out.

Reclin2Linkage (Implementation Class)

Description

The Reclin2Linkage class is the default implementation class that comes with the package on installation (*and metadata creation*), the implementation uses functions strictly from the **reclin2** package in R.

Details

Within the implementation, there are **three techniques** that the user can choose between for the linkage process, a **probabilistic**, **deterministic**, or **machine learning** process.

Probabilistic: Before any steps take place, pre-processing needs to occur to make sure we have all the information required for probabilistic linkage. This includes obtaining the blocking and matching criteria, and modifying the field names so that they appear the same in both datasets; apply linkage rules to field pairs such as name standardization, integer variance, substring, or alternative field values; and finally generating a list of blocking key pairs (*if we may have multiple vectors of blocking keys*).

Step 1 – Generate Pairs: Block on our first pair of blocking criteria, and then bind any additional blocking criteria if multiple vectors are provided (*via linkage rules*).

Step 2 – Compare Pairs: Keep a list of the matching criteria that will be used for comparing pairs, along with a list of key value pairs, where a comparison function defined by the user in the linkage metadata GUI will be applied to the values of that specific field, which will be used in the **comparator** field of the reclin2 **compare_pairs()** function.

Step 3 – Score Pairs: Create a formula using the matching criteria, and use it to make a prediction for scoring the pairs.

Step 4 – Select Pairs: Using the acceptance threshold stored in the metadata, either select the pairs that will be considered successfully linked, creating a column of logicals (*TRUE or FALSE*) that indicate whether that pair should link

or not. If no acceptance threshold is provided, all future steps are halted, and histograms of the scores are generated, with only certain values being returned.

Step 5 – Link Pairs: Link the pairs using the logical value obtained during step 4, keeping all the left and right dataset values, and scores, appending the stage name/pass name as a new column.

Step 6 – Return Values: Create a list and append data that is to be returned to it. This includes the **linked indices**; the **unlinked dataset pairs**; the **linked dataset pairs**; the **filtered data** (*data with the output fields we want to keep*); **threshold plots** and **threshold captions** if the user had defined a ground truth for the algorithm; and **performance measure variables** if the user had defined a ground truth for the algorithm. The list is then returned.

Deterministic: Before any steps take place, pre-processing needs to occur to make sure we have all the information required for deterministic linkage. This includes obtaining the blocking and matching criteria, and modifying the field names so that they appear the same in both datasets; apply linkage rules to field pairs such as name standardization, integer variance, substring, or alternative field values; and finally generating a list of blocking key pairs (*if we may have multiple vectors of blocking keys*).

Step 1 – Generate Pairs: Block on our first pair of blocking criteria, and then bind any additional blocking criteria if multiple vectors are provided (*via linkage rules*).

Step 2 – Compare Pairs: Keep a list of the matching criteria that will be used for comparing pairs, along with a list of key value pairs, where a comparison function defined by the user in the linkage metadata GUI will be applied to the values of that specific field, which will be used in the **comparator** field of the reclin2 **compare_pairs()** function.

Step 3 – Score Pairs: Pair scoring is simpler for deterministic linkage, as we need to fully match on all our matching criteria. Because of this, a match is given a score of **1** if a match is successful, and **0** otherwise.

Step 4 – Select Pairs: To select pairs in deterministic linkage, we need to make sure that the sum of all matching scores is equal to the number of matching criteria we are using. If it wavers by even a single point then the pair being selected is not considered a match.

Step 5 – Link Pairs: Link the pairs using the logical value obtained during step 4, keeping all the left and right dataset values, and scores, appending the stage name/pass name as a new column.

Step 6 – Return Values: Create a list and append data that is to be returned to it. This includes the **linked indices**; the **linked dataset pairs**; the **filtered data** (*data with the output fields we want to keep*). The list is then returned.

Machine Learning: Before any steps take place, pre-processing needs to occur to make sure we have all the information required for machine learning linkage. This includes obtaining the blocking and matching criteria, and modifying the field names so that they appear the same in both datasets; apply linkage rules to field pairs such as name standardization, integer variance, substring, or alternative field values; and finally generating a list of blocking key pairs (*if we may have multiple vectors of blocking keys*).

Step 1 – Generate Pairs: Block on our first pair of blocking criteria, and then bind any additional blocking criteria if multiple vectors are provided (*via linkage rules*).

Step 2 – Compare Pairs: Keep a list of the matching criteria that will be used for comparing pairs, along with a list of key value pairs, where a comparison function defined by the user in the linkage metadata GUI will be applied to the values of that specific field, which will be used in the **comparator** field of the reclin2 **compare_pairs()** function. With machine learning, we also obtain our ground truth variables here and add a column indicating whether ground truth matches.

Step 3 – Create an ML Model: We'll make a formula using the matching criteria and ground truth matches, using the formula to construct a machine learning model to be applied to our linkage pairs.

Step 4 – Apply ML Model to Pairs: Using the acceptance threshold stored in the metadata, either select the pairs that the model selected that exceed a certain probability, creating a column of logicals (*TRUE or FALSE*) that indicate whether that pair should link or not. If no acceptance threshold is provided, all future steps are halted, and histograms of the scores are generated, with only certain values being returned.

Step 5 – Link Pairs: Link the pairs using the logical value obtained during step 4, keeping all the left and right dataset values, and scores, appending the stage name/pass name as a new column.

Step 6 – Return Values: Create a list and append data that is to be returned to it. This includes the **linked indices**; the **unlinked dataset pairs**; the **linked dataset pairs**; the **filtered data** (*data with the output fields we want to keep*); **threshold plots** and **threshold captions** if the user had defined a ground truth for the algorithm; and **performance measure variables** if the user had defined a ground truth for the algorithm. The list is then returned.

Creating a Custom Linkage Implementation

Description

One of the features the main linkage function has is that the linkage implementations are dynamically called using the selected implementation in the database. If the user wishes to program their own implementation to be used, then the following steps should be followed to ensure that the class is set up, reachable, and returns the necessary information.

Details

To construct a custom linkage implementation that you can pair with the packages default class, follow the steps as follows…

Step 1 – Inheriting the Abstract Linkage Class: Inherit the abstract **LinkageMethod** class by creating an R6 class with your class, which may be named for example, **CustomClass**. You will use the **inherit** parameter within the R6Class object and reference the abstract class from this package.

Step 2 – Constructing Your Implementation Function: Within your class, you will reference the **run_iteration()** function, using the exact parameters from the abstract class. From here, you may now program your implementation as you wish, using whichever combinations of functions, packages, or objects that you may need. What is required however, is the expected output. The function should return the following in a list, with the parameter names as described, otherwise, your output may not be recorded successfully.

> **Linked Indices:** A vector of integers, containing the indices that were successfully linked. If no indices were successfully linked, then return NA. This should be stored in the return list as "**linked_indices**".
>
> **Unlinked Dataset Pairs:** A data frame of the unlinked dataset pairs before the 'link' step takes place. This should be stored in the return list as "**unlinked_dataset_pairs**".
>
> **Linked Dataset:** A data frame of the linked dataset pairs after the 'link' step takes place. This should be stored in the return list as "**linked_dataset**".
>
> **Performance Measure Variables:** If a ground truth was provided, and you were able to calculate the performance measure variables, they must be stored as a vector of **four values**, which are **TP**, **TN**, **FP**, **FN**, in that order. This should be stored in the return list as "**performance_measure_variables**".

Filtered Output Data: A data frame of the linked data, only containing the output fields defined by the user in the GUI. The output fields must match what is stored in the metadata exactly and must not contain any other fields besides what is defined by the user. This should be stored in the return list as "**output_linkage_df**".

Threshold Plots & Captions: If histograms were created during data linkage, they should be stored in a list while the captions should be a vector of character strings. The list of plots can have any variable name for storing the plots but words should be separated by **underscores** or **dashes** to make reading the PNG titles easier. The number of plots and captions MUST equal. The plots should be stored in the return list as "**threshold_plots**" while the individual plot captions should be stored in the return list as "**plot_captions**".

Step 3 – Adding the Implementation to the Metadata: Within the linkage metadata GUI, the user may progress to the **Linkage Methods** page and fill in the required information for adding their implementation. The **Implementation/Class Name** should match the name of the R class the user created EXACTLY, so if the user specified their class as being called **CustomClass**, that is what the user should supply to the input field. The linkage method is used to determine what type of linkage should be executed within your class. Because of this, if the user has multiple methods, they will need to add each individual method with the same class name.

Step 4 – Selecting the Custom Implementation: Once the implementation has been successfully added, it can now be selected when creating the individual passes of an algorithm, under the **Linkage Implementation** input selection.

Step 5 – Using The Custom Implementation: To use the class properly, the user must be sure to **source** their script containing the class prior to running linkage in the current session as the main linkage function will attempt to search for the class within the user's environment. If created, added, and sourced successfully, the function will call your implementation, passing in the required data as parameters.

linkage_metadata_ui.R

Description

linkage_metadata_ui.R is the shiny application that will allow users to make various changes to and run their linkage algorithms.

Details

The available specifications/pages are:

- Home
- Datasets
- Linkage Methods
- Testable Linkage Algorithms
- Archived Linkage Algorithms
- Published Linkage Algorithms
- View Linkage Iterations
- Modify Linkage Iterations
- Comparison Methods
- Acceptance Methods
- Ground Truth Variables
- Linkage Algorithm Output Fields
- Linkage Audits
- Run Algorithm(s)
- Regenerate Report

Home

Description

The home page provides users a with a descriptive summary of the common workflow from adding a new dataset to algorithm creation, while also providing them with the other accessible pages within the GUI and how to get from the start page to them.

Details

Three accordion tabs are provided on the home page, each of which are labeled with a **step number** which helps designate the average path for a user to take when creating an algorithm for the first time. The last tab provides the user with a way to get access to other pages.

Datasets

Description

The datasets page allows users to add or modify datasets to be used for linkage algorithms. A minimum of two datasets will need to exist before constructing an algorithm takes place.

Details

There are some global fields that are used to store information before fully inserting the information into the database so that users can verify that what they're uploading is correct. This information includes a data frame that contains the currently uploaded fields; a file path for the uploaded file when adding a new dataset; and a file path for the uploaded file when updating an existing dataset.

get_datasets()	This function performs a query that returns all the currently uploaded datasets (*enabled or not*), renaming the columns to be more human readable, and returning a data table that can be rendered in R Shiny.
read_dataset_columns()	This function will read the first row from the uploaded dataset and attempt to extract the column names, returning a vector of the columns if successful. This function can read in files of types: **csv**, **txt**, **sas7bdat**, **xlsx**, **xls**.
read_dataset_col_types()	This functional will read in the first 100 rows from the uploaded dataset and attempt to estimate/guess the field types of each column. This function can read in files of types: **csv**, **txt**, **sas7bdat**, **xlsx**, **xls**.
currently_added_datasets (RenderDataTable)	This output rendering function calls the **get_datasets()** function and renders the UI element with ID **currently_added_datasets** with the data table returned from said function.
selected_dataset_fields (RenderDataTable)	This output rendering function uses the selection the user made by identifying which row from the **currently_added_datasets** element was selected, and populating a table of the fields belonging to that selected dataset.
uploaded_dataset_fields (RenderDataTable)	This output rendering function uses the uploaded dataset fields obtained from the file uploaded by the user, rendering the UI element with ID **uploaded_dataset_fields** with a datatable.

Observe (Uploaded File & Fixed-Width Toggle)	Checks for whether the user uploaded an input file, or if they toggled the currently uploaded dataset as being fixed-width formatting. If fixed-width was toggled, the widths will be estimated and provided to the user for manual editing if required.
Observe Event (Uploaded Datasets Row Selected)	Checks which uploaded field the user selected and will pre-populate the fixed-width input field with the estimate (*or manually added width*).
Observe Event (Update Fixed-Width Button)	Obtains the field the user selected and changes the fixed-width value of the selected field to be the value obtained from the users input.
Observe Event (Toggle Dataset)	Obtains the row (*dataset*) the user selected and toggles the enabled value in the metadata to be either **0** or **1**. If **1** initially, it will be set to **0** which indicates that it is disabled, meaning that it can't be used for data linkage, and no algorithms can be created, modified, toggled so long as it is disabled. If **0** initially, it will be set to **1** which indicates that it is enabled, meaning it can be used normally.
Observe Event (Add Dataset File)	Allows the user to upload a file that changes the reactive value of the currently added dataset file. This results in the column names and types to be updated (*which also modifies the table of currently uploaded dataset fields*).
Observe Event (Update Dataset File)	Allows the user to replace the selected dataset with a new file which will replace the reactive uploaded file for updating. This results in the column names and types to be updated (*which also modifies the table of currently uploaded dataset fields*).
Observe (Add Dataset & Update Dataset File Paths)	Observes changes in the uploaded files for datasets being added or updated, this will render text output of the file that is currently uploaded which will help users see that they properly uploaded the file they selected.
Observe Event (Add Dataset)	Adds a new dataset to the database by taking the **dataset code**, **name**, **version**, **fixed-width toggle**, and **file** inputs from the user. Error handling is done to prevent uploading empty inputs, and preventing that two datasets exist at the same time with the same code. A transaction occurs during the insertion process since we're inserting into multiple tables and are also making potentially many insert queries which can be disastrous if a crash or error occurs before we finish inserting all the data. Once the data is inserted, we will reset the input elements and re-render all tables.

Observe (Added Datasets Row Selected)	Observes for when a user selects a row (*existing dataset*) in the table of currently added datasets, which will pre-populate the input fields with the database information for easy updating.
Observe Event (Update Dataset)	Updates the existing currently selected dataset in the database by taking the **dataset code**, **name**, **version**, **fixed-width toggle**, and **file** inputs from the user. Error handling is done to prevent updating with empty inputs, and preventing that two datasets exist at the same time with the same code. Once the data is updated, we will reset the input elements and re-render all tables.

Linkage Methods

Description

The linkage methods page allows the user to submit the name and method their own custom linkage class (*written and scripted outside of the UI*). The linkage method will be usable when choosing the implementation of each individual pass of an algorithm.

Details

get_linkage_methods()	This function performs a query which returns all the currently inserted linkage methods, renaming the columns to be more human readable, and returning a data table that can be rendered in R Shiny.
currently_added_linkage_methods (RenderDataTable)	This rendering function calls the **get_linkage_methods()** function to render a data table of the linkage methods.
Observe Event (Add Linkage Method)	Adds a new linkage method to the database by taking the users inputs of the **implementation name**, **description**, **technique label**, and **version**. Error handling is performed to ensure that no two linkage methods share the same implementation name and label.

Testable Linkage Algorithms

Description

The testable linkage algorithms page allows the user to select the **left** and **right** dataset that they want to add, modify, view the algorithms of. From this page, you can run the algorithms, regenerate reports, while also modifying the passes, ground truth, and output fields of each algorithm.

Details

Observe Event (Linkage Algorithms -> View Linkage Iterations)	Observes for when the user presses the button that brings the user to the view linkage iterations page, this function will pre-populate some global variable information on the view linkage iteration page before changing the tabs for the user.
get_left_datasets() [linkage algorithms]	This function will perform a query to get the dataset names and IDs from the database, using the query result to create a lookup table of **names** that map to **ID** values. Returns a select input UI element.
get_right_datasets() [linkage algorithms]	This function will perform a query to get the dataset names and IDs from the database, using the query result to create a lookup table of **names** that map to **ID** values. Returns a select input UI element.
Linkage Algorithm Left Dataset Input (RenderUI)	Renders a UI element by using the output from the **get_left_datasets_linkage_algorithms()** function.
Linkage Algorithm Right Dataset Input (RenderUI)	Renders a UI element by using the output from the **get_right_datasets_linkage_algorithms()** function.
get_linkage_algorithms()	Performs a query using the two selected dataset inputs (*using the dataset IDs*) and renders a data table of all the currently testable linkage algorithms belonging to this dataset pair, renaming columns to make them more human readable.
Currently Added Linkage Algorithms (RenderDataTable)	Renders the table of all currently added linkage algorithm by using the output from the **get_linkage_algorthms()** function.
Observe Event (Add Linkage Algorithm)	Adds a blank algorithm ready for modification by using the provided inputs of an **algorithm name**, and a **left** and **right dataset ID**. Error handling is performed to make sure no inputs are blank and that an algorithm with the same name isn't already being tested. Once validated, an insert query into the **linkage_algorithms** table is performed, inputs are reset, and tables are re-rendered.

Observe (Currently Added Linkage Algorithms Row Selected)	Checks for if a user selected one of the currently testable algorithms, which will pre-populate the algorithm name update input field for easier updating.
Observe Event (Update Linkage Algorithm)	Updates the currently selected linkage algorithm by using the provided inputs of an **updated algorithm name**, and a **left** and **right dataset ID**. Error handling is performed to make sure no inputs are blank and that an algorithm with the same name isn't already being tested. Once validated, an insert query into the **linkage_algorithms** table is performed, inputs are reset, and tables are re-rendered.
Observe Event (Toggle Algorithm as Default)	Obtains the row (*algorithm*) the user selected and toggles the 'enabled as default algorithm' value in the metadata to be either **0** or **1**. If **1** initially, it will be set to **0** which indicates that it is disabled, meaning that it is not considered the default algorithm, and that it will not contain additional information in the appendix when generating a report. If **0** initially, it will be set to **1** which indicates that it is enabled, meaning we considered this algorithm as our current best while testing, containing additional information in the appendix about the other non-enabled algorithms.
Observe Event (Toggle Algorithm for Testing)	Obtains the row (*algorithm*) the user selected and toggles the 'enabled for testing' value in the metadata to be either **0** or **1**. If **1** initially, it will be set to **0** which indicates that it is disabled, meaning that its information will not appear in the report generated by the default algorithm. If **0** initially, it will be set to **1** which indicates that it is enabled, meaning we considered this algorithm for testing and that additional information will appear in the report generated by the default algorithm.
Observe Event (Linkage Algorithms -> Ground Truth Variables)	Observes for when the user presses the button that brings the user to the modify ground truth variables page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Linkage Algorithms -> Saved Performance Measures)	Observes for when the user presses the button that brings the user to the saved performance measures page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Linkage Algorithms -> Algorithm Outputs)	Observes for when the user presses the button that brings the user to the modify algorithm outputs page, this function will pre-populate some global variable information on the page before changing the tabs for the user.

Observe Event (Linkage Algorithms -> Run Algorithms)	Observes for when the user presses the button that brings the user to the run algorithms page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Linkage Algorithms -> Alternative Run Algorithms)	Observes for when the user presses the button that brings the user to the run algorithms page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Linkage Algorithms -> Regenerate Reports)	Observes for when the user presses the button that brings the user to the regenerate reports page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Archive Algorithm)	Brings up a modal dialog box for confirming whether the user wants to archive the selected algorithm or cancel their selection.
Observe Event (Archive Algorithm Confirm)	If the user confirms to archiving the selected algorithm, the modal will close, and the algorithm will have an update query performed on it which will toggle the **archived** value to **1** which means it will not be considered as being testable or publishable but can be restored from the archived algorithms page.
Observe Event (Publish Algorithm)	Brings up a modal dialog box for confirming whether the user wants to publish the selected algorithm or cancel their selection.
Observe Event (Publish Algorithm Confirm)	If the user confirms to publishing the selected algorithm, the modal will close, and the algorithm will have an update query performed on it which will toggle the **published** value to **1** which means it will not be considered as being testable or archivable but can be run and unpublished from the published algorithms page.
Observe Event (Linkage Algorithms -> Archived Algorithms)	Observes for when the user presses the button that brings the user to the archived algorithms page, this function will pre-populate some global variable information on the page before changing the tabs for the user.
Observe Event (Linkage Algorithms -> Published Algorithms)	Observes for when the user presses the button that brings the user to the published algorithms page, this function will pre-populate some global variable information on the page before changing the tabs for the user.

Archived Linkage Algorithms

Description

The archived linkage algorithms page allows users to view and restore the algorithms they decided were unfit for publication or current testing purposes. Selecting an algorithm will allow the user to briefly look over the algorithm details, but will be unable to modify anything until the user restores the algorithm, bringing it back to the testing page.

Details

Global variables are used to keep track of the currently selected **dataset pair** that the user selected on the **Testable Linkage Algorithms** page, as well as a character string for which page to return to when pressing the back button.

Observe Event (Archived Algorithms Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_archived_algorithms()	This function will perform a query to get all the currently archived linkage algorithms that pertain to the current dataset pair. Renaming columns so that they are more human readable.
Archived Linkage Algorithms (RenderDataTable)	This rendering function will take the output from **get_archived_linkage_algorithms()** and render a data table with the output.
Observe Event (Archived Linkage Algorithms Row Selected)	Observes the currently selected row (*algorithm*) in the table of archived algorithms and renders a data table containing the pass-level information.
Observe Event (Restore Linkage Algorithm)	Will attempt to restore the selected archived linkage algorithm by performing an update query on the algorithm, toggling the **archived** value to **0** from **1**, so long as that there isn't a testable or published algorithm that is using the same name.

Published Linkage Algorithms

Description

The published linkage algorithms page allows users to view and unpublish the algorithms they decided were fit for publication or current testing purposes. Selecting an algorithm will allow the user to briefly look over the algorithm details but will be unable to modify anything until the user unpublishes the algorithm, bringing it back to the testing page.

Details

Global variables are used to keep track of the currently selected **dataset pair** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, and computer volumes for choosing an output directory via **shinyDirChoose()**.

Selected Published Algorithms Output Directory (RenderText)	Initially renders the text output with "No Folder Has Been Chosen".
Observe Event (Published Algorithms Output Directory)	Observed the selected output directory and will render the output text to reflect the directory to the user for ensuring that it was correctly selected.
Observe Event (Published Algorithms Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_published_algorithms()	This function will perform a query to get all the currently published linkage algorithms that pertain to the current dataset pair. Renaming columns so that they are more human readable.
Published Linkage Algorithms (RenderDataTable)	This rendering function will take the output from **get_published_linkage_algorithms()** and render a data table with the output.
Observe Event (Published Linkage Algorithms Row Selected)	Observes the currently selected row (*algorithm*) in the table of published algorithms and renders a data table containing the pass-level information.
Observe Event (Unpublish Linkage Algorithm)	Will attempt to restore the selected published linkage algorithm by performing an update query on the algorithm, toggling the **published** value to **0** from **1**, so long as that there isn't a testable algorithm that is using the same name.
Observe Event (Run Published Linkage Algorithms)	Will attempt to run all the currently published linkage algorithms, using the results returned from the main linkage function by saving each of the algorithms return data as an **.Rdata** file in the supplied output directory.

View Linkage Iterations (Passes)

Description

The view linkage iterations page allows users to view all of the currently added passes for the currently selected algorithm at the top of page, which includes the each pass along with its order, method/technique, blocking and matching criteria, acceptance rules, modification date, modification author, and whether or not it is enabled.

Details

Global variables are used to keep track of the currently selected **dataset pair** and **algorithm ID** that the user selected on the **Testable Linkage Algorithms** page, and a character string for which page to return to when pressing the back button.

Observe Event (View Linkage Iterations Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_linkage_iterations_view()	Performs a query on the metadata using the passed algorithm ID, obtaining all iterations belonging to the corresponding algorithm. Each row of the table returned being an individual iteration which includes the matching and blocking criteria, acceptance thresholds, linkage method & technique, who last modified it, and last modified date.
Currently Added Linkage Iterations (RenderDataTable)	Renders a data table using the **output** from the **get_linkage_iterations_view()** function.
get_linkage_iterations_add_existing()	Performs a query on the metadata using the passed left and right dataset IDs, obtaining all previously used iterations/passes belonging to the dataset pair. Includes all iteration information like the previous function and allows the user to select one of the existing iterations as a baseline/starting point.
Previously Used Iterations (RenderDataTable)	Renders a data table using the **output** from the **get_linkage_iterations_add_existing()** function.
Observe Event (Add New Linkage Iteration)	Brings the user to the “Modify Linkage Iterations” page, allowing the user to create a new linkage iteration from scratch.
Observe Event (Add Existing Linkage Iteration)	Uses the selected row of the existing iterations and brings the user to the “Modify Linkage Iterations”, pre-populating the user inputs with the selected iterations information which may be used as a starting point for creating the new iteration.

Observe Event (Toggle Linkage Iteration)	Obtains the row (*iteration*) the user selected and toggles the 'enabled' value in the metadata to be either **0** or **1**. If **1** initially, it will be set to **0** which indicates that it is disabled, meaning that the iteration will not be used when running the algorithm it belongs to. If **0** initially, it will be set to **1** which indicates that it will be used for data linkage when the algorithm it belongs to is ran.
Observe Event (Modify Linkage Iteration)	Brings the user to the "Modify Linkage Iterations" page using the selected row of currently added iterations for the algorithm the user is modifying. Pre-populating the inputs for the user to modify to their liking.

Modify Linkage Iterations (Passes)

Description

The modify linkage iterations page allows users to create the individual passes that make up an algorithm to their liking, choosing the order they run in, implementations they'll use, blocking and matching criteria, and acceptance thresholds.

Details

Global variables are used to keep track of the currently selected **dataset pair** and **algorithm ID** that the user selected on the **Testable Linkage Algorithms** page, a reactive value for the selected standardization file path, and a character string for which page to return to when pressing the cancel or submit button.

Additionally, global variables are used to store *prepared* data which is the left and right blocking keys to add, the left and right matching keys to add, and any comparison or linkage rules that may belong to each pair.

Get Left Dataset Blocking Fields To Add (Function)	Performs a query using the provided **left dataset ID**, will create a select input UI element for the available left fields that we can block on when adding a pair of fields.
Get Right Dataset Blocking Fields To Add (Function)	Performs a query using the provided **right dataset ID**, will create a select input UI element for the available right fields that we can block on when adding a pair of fields.
Get Left Dataset Blocking Fields To Update (Function)	Performs a query using the provided **left dataset ID**, will create a select input UI element for the available left fields that we can block on when updating a pair of fields.

Get Right Dataset Blocking Fields To Update (Function)	Performs a query using the provided **right dataset ID**, will create a select input UI element for the available right fields that we can block on when updating a pair of fields.
Add Left Blocking Field Input (RenderUI)	Calls the **get_left_dataset_blocking_fields_to_add()** function and renders the **selectInput UI** element**.**
Add Right Blocking Field Input (RenderUI)	Calls the **get_right_dataset_blocking_fields_to_add()** function and renders the **selectInput UI** element**.**
Update Left Blocking Field Input (RenderUI)	Calls the **get_left_dataset_blocking_fields_to_update()** function and renders the **selectInput UI** element**.**
Update Right Blocking Field Input (RenderUI)	Calls the **get_right_dataset_blocking_fields_to_update()** function and renders the **selectInput UI** element**.**
Get Left Dataset Matching Fields To Add (Function)	Performs a query using the provided **left dataset ID**, will create a select input UI element for the available left fields that we can match on when adding a pair of fields.
Get Right Dataset Matching Fields To Add (Function)	Performs a query using the provided **right dataset ID**, will create a select input UI element for the available right fields that we can match on when adding a pair of fields.
Get Left Dataset Matching Fields To Update (Function)	Performs a query using the provided **left dataset ID**, will create a select input UI element for the available left fields that we can match on when updating a pair of fields.
Get Right Dataset Matching Fields To Update (Function)	Performs a query using the provided **right dataset ID**, will create a select input UI element for the available right fields that we can match on when updating a pair of fields.
Add Left Matching Field Input (RenderUI)	Calls the **get_left_dataset_matching_fields_to_add()** function and renders the **selectInput UI** element**.**
Add Right Matching Field Input (RenderUI)	Calls the **get_right_dataset_matching_fields_to_add()** function and renders the **selectInput UI** element**.**
Update Left Matching Field Input (RenderUI)	Calls the **get_left_dataset_matching_fields_to_update()** function and renders the **selectInput UI** element**.**
Update Right Matching Field Input (RenderUI)	Calls the **get_right_dataset_matching_fields_to_update()** function and renders the **selectInput UI** element**.**
Observe Event (Remove Iteration Acceptance Rule)	Resets the global variable associated with the prepared acceptance rule with an **NA** value and renders the selected acceptance rule text box to be empty.
Observe Event (Remove Blocking Linkage Rule)	Resets the global variable associated with the prepared blocking linkage rule when adding a blocking pair with an **NA** value and renders the selected blocking linkage rule text box to be empty.

Observe Event (Remove Blocking Linkage Rule Update)	Resets the global variable associated with the prepared blocking linkage rule when updating a blocking pair with an **NA** value and renders the selected blocking linkage rule text box to be empty.
Observe Event (Remove Matching Linkage Rule)	Resets the global variable associated with the prepared matching linkage rule when adding a matching pair with an **NA** value and renders the selected matching linkage rule text box to be empty.
Observe Event (Remove Matching Linkage Rule Update)	Resets the global variable associated with the prepared matching linkage rule when updating a matching pair with an **NA** value and renders the selected matching linkage rule text box to be empty.
Observe Event (Remove Matching Comparison Rule)	Resets the global variable associated with the prepared matching comparison rule when adding a matching pair with an **NA** value and renders the selected matching comparison rule text box to be empty.
Observe Event (Remove Matching Comparison Rule Update)	Resets the global variable associated with the prepared acceptance rule with an **NA** value and renders the selected acceptance rule text box to be empty.
Observe Event (Remove Standardization Dataset)	Resets the global variable associated with the prepared standardization dataset ID with an **NA** value and renders the selected standardization dataset text box to be empty.
get_linkage_methods_to_add()	Performs a query of all the currently available linkage methods & techniques which will allow the user to choose one via a select input UI element.
Add Iteration Linkage Method Input (RenderUI)	Calls the **get_linkage_methods_to_add()** function and uses the output to render a **selectInput UI** element.
get_blocking_keys_to_add()	Uses the prepared left and right blocking keys we have prepared as global variables, and formats them (*along with their linkage rules*) into an easy-to-read table. This involves querying the IDs and turning them into the readable field names and rules.
Add Blocking Variables Table (RenderDataTable)	Calls the **get_blocking_keys_to_add()** function and uses the output to render a **Data Table UI** element.
get_matching_keys_to_add()	Uses the prepared left and right matching keys we have prepared as global variables, and formats them (*along with their linkage rules and/or comparison rules*) into an easy-to-read table. This involves querying the IDs and turning them into the readable field names and rules.
Add Matching Variables Table (RenderDataTable)	Calls the **get_matching_keys_to_add()** function and uses the output to render a **Data Table UI** element.

Observe Event (Return From Add Iterations)	Allows the user to return to the "View Linkage Iterations" page without saving their changes for the currently selected iteration. All progress will be lost, and the user will be brought back to the view page.
Observe Event (Prepare Standardization Dataset)	When the user clicks the pencil icon next to the standardization dataset text box, a modalDialog is brought up which will prompt the user to either upload a new dataset file for use, or they may select an existing dataset (**.Rds**) file that is stored in the packages AppData.
Observe Event (Upload Standardization File)	Allows the users to select a file on their computer which they would like to use as their standardization dataset for this iteration.
Observe (Uploaded Standardization File)	Observes the uploaded files, and takes the base name of the file to render in a text output UI element to show the user which file they uploaded.
Observe Event (Select Uploaded Standardization File)	If the user wants to use an uploaded file as the standardization dataset for this iteration, first, the uploaded file must be verified by checking if it has the columns "**unique**" and "**common**". If so, the dataset is read in, saved as a **.Rds** file and stored in the **autolink** system files.
Observe Event (Select Standardization File)	If the user wants to use an already uploaded file/file stored in the **autolink** system files, they can select an existing dataset in the provided select input UI element.
Modify Iteration Get Acceptance Rule Inputs (Function)	Performs a query using the provided acceptance method ID and dynamically generates a list of UI elements with the provided label name which will create text inputs of the input parameters under the corresponding acceptance method ID.
Modify Iteration Get Comparison Rule Inputs (Function)	Performs a query using the provided comparison method ID and dynamically generates a list of UI elements with the provided label name which will create text inputs of the input parameters under the corresponding comparison method ID.
Observe Event (Prepare Iteration Acceptance Rule)	When the user clicks the pencil icon next to the acceptance rule text box, a modalDialog is brought up which will prompt the user to select an acceptance method of their choice that they'd like to use, and then type in numeric input for each of the provided parameters.
Add Acceptance Method Iteration (RenderDataTable)	Calls the **get_acceptance_methods_and_parameters()** function, using the output to render a **Data Table UI** element.

Observe Event (Add Acceptance Method Iteration Rows Selected)	Observes the row (*acceptance method*) that the user selected in the **Add Acceptance Method Iteration** table and calls the **modify_iteration_get_acceptance_rule_inputs**() function to render **selectInput UI** elements.
Observe Event (Prepare Iteration Acceptance Rule to Add)	Selects the acceptance rule the user wanted, verifying that the inputs are not blank, and performs a query to check if the provided acceptance rule already has been added. If it has, the existing acceptance rule ID is prepared, if not, an insert query is performed, and the newly added acceptance rule ID is prepared.
Observe Event (Add Linkage Iteration to Add Acceptance Methods)	If the user would like to add and use their own acceptance method (*if it does not already exist*), then when no row is selected within the modalDialog, the blue button will bring the user to the "Add Acceptance Methods" page.
Observe Event (Prepare Blocking Variables)	Takes the input values of the **left** and **right blocking keys** (*plus any selected linkage rules*) and will add them to our global prepared values. Error handling prevents empty inputs from being prepared, or existing pairs to be added again. The data table of blocking keys is re-rendered to show what is currently prepared.
Observe (Add Blocking Variables Table Rows Selected)	Observes the row (*pair of blocking keys and linkage rules*) the user selected of the prepared blocking keys, which will have the selectInput UI elements and linkage rules pre-populated with what is currently prepared.
Observe Event (Prepare Blocking Variables Update)	Takes the input values of the **left** and **right blocking keys** (*plus any selected linkage rules*) and will update the selected row of our global prepared values. Error handling prevents empty inputs from being prepared, or existing pairs to be added again. The data table of blocking keys is re-rendered to show what is currently prepared after the update.
Observe Event (Drop Blocking Variables)	Drops the selected pair of blocking keys (*plus any linkage rules*) from the global prepared blocking keys. The data table of blocking keys is re-rendered to show what is currently prepared after dropping the keys.
Observe Event (Prepare Blocking Linkage Rule)	When the user clicks the pencil icon next to the add blocking keys linkage rules text box, a modalDialog is brought up which will prompt the user to fill in inputs for any number of the available linkage rules and then submit the inputs for preparing the rules.

Observe Event (Prepare Blocking Linkage Rule Update)	When the user clicks the pencil icon next to the update blocking keys linkage rules text box, a modalDialog is brought up which will prompt the user to fill in inputs for any number of the available linkage rules and then submit the inputs for preparing the rules.
Add and Select Linkage Rule (Function)	Dynamically uses the provided global id parameter which is used to collect the corresponding inputs, and use them to either use an existing linkage rule ID (*if the combination of inputs already exists*), or an insertion query will insert the linkage rules into the database, and the newly created linkage rule ID will be used. The corresponding global IDs value will be prepared after obtaining the necessary linkage rule ID.
Observe Event (Add Linkage Rule Blocking)	Calls the **add_and_select_linkage_rule()** function, passing the global ID of the blocking linkage rule that is to be added.
Observe Event (Update Linkage Rule Blocking)	Calls the **add_and_select_linkage_rule()** function, passing the global ID of the blocking linkage rule that is to be updated.
Observe Event (Prepare Matching Variables)	Takes the input values of the **left** and **right matching keys** (*plus any selected linkage and/or comparison rules*) and will add them to our global prepared values. Error handling prevents empty inputs from being prepared, or existing pairs to be added again. The data table of matching keys is re-rendered to show what is currently prepared.
Observe (Add Matching Variables Table Rows Selected)	Observes the row (*pair of matching keys, linkage and/or comparison rules*) the user selected of the prepared matching keys, which will have the selectInput UI elements linkage and/or comparison rules pre-populated with what is currently prepared.
Observe Event (Prepare Matching Variables Update)	Takes the input values of the **left** and **right matching keys** (*plus any selected linkage and/or comparison rules*) and will update the selected row of our global prepared values. Error handling prevents empty inputs from being prepared, or existing pairs to be added again. The data table of matching keys is re-rendered to show what is currently prepared after the update.
Observe Event (Drop Matching Variables)	Drops the selected pair of matching keys (*plus any linkage and/or comparison rules*) from the global prepared matching keys. The data table of matching keys is re-rendered to show what is currently prepared after dropping the keys.

Observe Event (Prepare Matching Linkage Rule)	When the user clicks the pencil icon next to the add matching keys linkage rules text box, a modalDialog is brought up which will prompt the user to fill in inputs for any number of the available linkage rules and then submit the inputs for preparing the rules.
Observe Event (Prepare Matching Linkage Rule Update)	When the user clicks the pencil icon next to the update matching keys linkage rules text box, a modalDialog is brought up which will prompt the user to fill in inputs for any number of the available linkage rules and then submit the inputs for preparing the rules.
Observe Event (Add Linkage Rule Matching)	Calls the **add_and_select_linkage_rule()** function, passing the global ID of the matching linkage rule that is to be added.
Observe Event (Update Linkage Rule Matching)	Calls the **add_and_select_linkage_rule()** function, passing the global ID of the matching linkage rule that is to be updated.
Observe Event (Prepare Matching Comparison Rule)	When the user clicks the pencil icon next to the add comparison rule text box, a modalDialog is brought up which will prompt the user to select a comparison method of their choice that they'd like to use, and then type in numeric input for each of the provided parameters.
Add Comparison Method Iteration (RenderDataTable)	Calls the **get_comparison_methods_and_parameters()** function, using the output to render a **Data Table UI** element.
Observe Event (Add Comparison Method Iteration Rows Selected)	Observes the row (*comparison method*) that the user selected in the **Add Comparison Method Iteration** table and calls the **modify_iteration_get_comparison_rule_inputs**() function to render **selectInput UI** elements.
Observe Event (Prepare Iteration Comparison Rule to Add)	Selects the comparison rule the user wanted, verifying that the inputs are not blank, and performs a query to check if the provided comparison rule already has been added. If it has, the existing comparison rule ID is prepared, if not, an insert query is performed, and the newly added comparison rule ID is prepared.
Observe Event (Add Linkage Iteration to Add Comparison Methods)	If the user would like to add and use their own comparison method (*if it does not already exist*), then when no row is selected within the modalDialog, the blue button will bring the user to the "Add Comparison Methods" page.
Observe Event (Prepare Matching Comparison Rule Update)	When the user clicks the pencil icon next to the update comparison rule text box, a modalDialog is brought up which will prompt the user to select a comparison method of their choice that they'd like to use, and then type in numeric input for each of the provided parameters.

Update Comparison Method Iteration (RenderDataTable)	Calls the **get_comparison_methods_and_parameters()** function, using the output to render a **Data Table UI** element.
Observe Event (Update Comparison Method Iteration Rows Selected)	Observes the row (*comparison method*) that the user selected in the **Add Comparison Method Iteration** table and calls the **modify_iteration_get_comparison_rule_inputs**() function to render **selectInput UI** elements.
Observe Event (Prepare Iteration Comparison Rule to Add Update)	Selects the comparison rule the user wanted, verifying that the inputs are not blank, and performs a query to check if the provided comparison rule already has been added. If it has, the existing comparison rule ID is prepared, if not, an insert query is performed, and the newly added comparison rule ID is prepared.
Observe Event (Add Linkage Iteration to Add Comparison Methods Update)	If the user would like to add and use their own comparison method (*if it does not already exist*), then when no row is selected within the modalDialog, the blue button will bring the user to the "Add Comparison Methods" page.
Observe Event (Save Iteration)	When the user goes to save the information, all the prepared global variables are verified, and we check whether the user is making an update to an existing algorithm or adding new information. If the information is new, make sure the iteration name is not being used, re-order the iterations based on the priority of the new iteration, perform a query that creates a new linkage iteration, then use that **iteration ID** to add the blocking keys and matching keys. If the iteration already exists, then delete/drop all information that belonged to the previous iteration, and re-insert the updated data, blocking keys, and matching keys under the **iteration ID** that it previously used. Afterwards, the user is brought back to the "View Linkage Iterations" page.
Observe Event (Preview Algorithm)	For users that want to preview their iterations, they can save the iteration and run it immediately to get information related to acceptance rules. This will perform the same checks in the **Save Iteration** event before moving the user to the "Run Algorithm(s)" Page.

Comparison Methods (*and Parameters*)

Description

The comparison methods page is accessible from the *Modify Linkage Iterations* page by pressing the blue "**Create Comparison Method**" button within the add comparison rule (*green pencil*) button. Within this page, you can add or modify comparison methods, including the method name, description, and parameters.

Details

Global variables are used to keep track of the page to return to, the prepared parameters and descriptions to add, and as well as the prepared parameters and descriptions to update.

Observe Event (Comparison Methods Back)	Observes for when the user presses the back button that brings the user back to the "Modify Linkage Iterations" page.
Get Comparison Methods and Parameters (Function)	Performs a query on the metadata that obtains all the currently created comparison methods, their descriptions, and the parameter key(s) that belong(s) to each. Returning the result as a DT table.
Currently Added Comparison Methods and Parameters (RenderDataTable)	Calls the **get_comparison_methods_and_parameters()** function, using the result to render a **Data Table UI** element.
Get Comparison Parameters to Add (Function)	Uses the global prepared list of parameter keys and descriptions that are being added to construct a DT table.
Comparison Parameters to Add (RenderDataTable)	Calls the **get_comparison_parameters_to_add()** function, using the result to render a **Data Table UI** element.
Observe Event (Prepare Comparison Method Parameters to Add)	Prepares the provided parameter key and description input values by verifying that they don't already exist for the comparison method that is being added.
Observe (Comparison Parameters to Add Rows Selected)	Observes the row (*parameter key and description pair*) the user selected and will pre-populate the update input fields for easier updating.
Observe Event (Update Prepared Comparison Method Parameters to Add)	Updates the selected prepared values with the provided updated parameter key and description input values by verifying that they don't already exist for the comparison method that is being added.
Observe Event (Drop Prepared Comparison Method Parameters to Add)	Drops the selected pair of prepared values from the global variables.

Observe Event (Add Comparison Method and Parameters)	Takes all input values including the comparison method name, description, and its individual parameters, verifying that the provided values don't result in a duplicate comparison method, finally performing an insert query to insert the data.
Get Comparison Parameters to Update (Function)	Uses the global prepared list of parameter keys and descriptions that are being updated to construct a DT table.
Comparison Parameters to Update (RenderDataTable)	Calls the **get_comparison_parameters_to_update()** function, using the result to render a **Data Table UI** element.
Observe (Currently Added Comparison Methods and Parameters Rows Selected)	Observes the currently selected row (*comparison method*) and pre-populates the general information and parameter specific information to allow for easier updating.
Observe (Comparison Parameters to Update Rows Selected)	Observes the row (*parameter key and description pair*) the user selected and will pre-populate the update input fields for easier updating.
Observe Event (Update Comparison Method Parameters to Update)	Updates the selected prepared values with the provided updated parameter key and description input values by verifying that they don't already exist for the comparison method that is being added.
Observe Event (Update Comparison Method and Parameters)	Takes all the updated input values including the comparison method name, description, and its individual parameters, verifying that the provided values don't result in a duplicate comparison method, finally performing an update query to replace the previous data with the updated information.

Acceptance Methods (*and Parameters*)

Description

The acceptance methods page is accessible from the *Modify Linkage Iterations* page by pressing the blue "**Create Acceptance Method**" button within the add acceptance rule (*green pencil*) button. Within this page, you can add or modify acceptance methods, including the method name, description, and parameters.

Details

Global variables are used to keep track of the page to return to, the prepared parameters and descriptions to add, and as well as the prepared parameters and descriptions to update.

Observe Event (Acceptance Methods Back)	Observes for when the user presses the back button that brings the user back to the "Modify Linkage Iterations" page.
Get Acceptance Methods and Parameters (Function)	Performs a query on the metadata that obtains all the currently created acceptance methods, their descriptions, and the parameter key(s) that belong(s) to each. Returning the result as a DT table.
Currently Added Acceptance Methods and Parameters (RenderDataTable)	Calls the **get_acceptance_methods_and_parameters()** function, using the result to render a **Data Table UI** element.
Get Acceptance Parameters to Add (Function)	Uses the global prepared list of parameter keys and descriptions that are being added to construct a DT table.
Acceptance Parameters to Add (RenderDataTable)	Calls the **get_acceptance_parameters_to_add()** function, using the result to render a **Data Table UI** element.
Observe Event (Prepare Acceptance Method Parameters to Add)	Prepares the provided parameter key and description input values by verifying that they don't already exist for the acceptance method that is being added.
Observe (Acceptance Parameters to Add Rows Selected)	Observes the row (*parameter key and description pair*) the user selected and will pre-populate the update input fields for easier updating.
Observe Event (Update Prepared Acceptance Method Parameters to Add)	Updates the selected prepared values with the provided updated parameter key and description input values by verifying that they don't already exist for the acceptance method that is being added.
Observe Event (Drop Prepared Acceptance Method Parameters to Add)	Drops the selected pair of prepared values from the global variables.
Observe Event (Add Acceptance Method and Parameters)	Takes all input values including the acceptance method name, description, and its individual parameters, verifying that the provided values don't result in a duplicate acceptance method, finally performing an insert query to insert the data.
Get Acceptance Parameters to Update (Function)	Uses the global prepared list of parameter keys and descriptions that are being updated to construct a DT table.
Acceptance Parameters to Update (RenderDataTable)	Calls the **get_acceptance_parameters_to_update()** function, using the result to render a **Data Table UI** element.

Observe (Currently Added Acceptance Methods and Parameters Rows Selected)	Observes the currently selected row (*acceptance method*) and pre-populates the general information and parameter specific information to allow for easier updating.
Observe (Acceptance Parameters to Update Rows Selected)	Observes the row (*parameter key and description pair*) the user selected and will pre-populate the update input fields for easier updating.
Observe Event (Update Acceptance Method Parameters to Update)	Updates the selected prepared values with the provided updated parameter key and description input values by verifying that they don't already exist for the acceptance method that is being added.
Observe Event (Update Acceptance Method and Parameters)	Takes all the updated input values including the acceptance method name, description, and its individual parameters, verifying that the provided values don't result in a duplicate comparison method, finally performing an update query to replace the previous data with the updated information.

Ground Truth Variables

Description

The ground truth variables page allows users to select the field that appears in both datasets that act to determine whether our linked pairs are true or false matches.

Details

Global variables are used to keep track of the currently selected **dataset pair** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, and a value for the currently selected comparison that would be added with the ground truth inputs.

Observe Event (Modify Ground Truth Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_ground_truth_variables()	This function performs a query that will obtain the saved ground truth variables for this pair of datasets, renaming the columns to make them more human readable. Returns a data table that is used for rendering.
Currently Added Ground Truth Variables (RenderDataTable)	Will render a data table by using the output from the **get_ground_truth_variables()** function.
left_ground_truth_fields()	This function will perform a query to get the left dataset field names and IDs from the database, using the query result to create a lookup table of **names** that map to **ID** values. Returns a select input UI element.

right_ground_truth_fields()	This function will perform a query to get the right dataset field names and IDs from the database, using the query result to create a lookup table of **names** that map to **ID** values. Returns a select input UI element.
Ground Truth Left Field Input (RenderUI)	Uses the select input UI element from the **left_dataset_ground_truth_fields()** function to render the UI.
Ground Truth Right Field Input (RenderUI)	Uses the select input UI element from the **right_dataset_ground_truth_fields()** function to render the UI.
Comparison Rule Events	The comparison rule events involve bringing up a dialog modal with the available comparison methods. Once a method is selected, the user will be prompted to fill in the input values of the selected method (*for each parameter*) and then they may select the comparison method to use. The events also allow the user to cancel the selected comparison method, or to add a new comparison method from within the modal.
Observe Event (Add Ground Truth)	The selected left dataset field, right dataset field, and optional comparison rule will be used in an insert query which will apply the ground truth pair on a **dataset pair-wise** insertion, meaning this change will occur for all other algorithms under the same left and right dataset.
Observe Event (Drop Ground Truth)	The selected ground truth pair will be dropped by a delete query using the selected pairs **parameter ID**. Since ground truth fields are **dataset pair-wise**, then this ground truth will drop for all algorithms underneath the selected left and right dataset.

Linkage Algorithm Output Fields

Description

The linkage algorithms output fields page allows users to select the field that will appear in the output results after linkage takes place.

Details

Global variables are used to keep track of the currently selected **algorithm ID** and **left dataset ID** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, and a reactive value for the optional standardization file that the user may upload.

Observe Event (Linkage Algorithm Output Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_algorithm_output_fields()	This function performs a query that will obtain the added output fields for this specific algorithm, renaming the columns to be more human readable. Returning a data table of the output fields.
Currently Added Algorithm Output Fields (RenderDataTable)	Will render the data table of currently added output fields for this algorithm by calling the **get_algorithm_output_fields()** function.
Linkage Algorithm Output Field Input (RenderUI)	Blankly renders in the UI for the field type specific input.
Observe Event (Linkage Algorithm Output Field Type)	Based on the field type selected by the user via select input, the rendered UI can be of **three different types**. **Type 1:** Generic input, which requires just the original field, this includes field types like year cutoffs, age cutoffs, length of name, number of names, etc. **Type 2:** Derived age requires two fields instead of the usual one, this involves a birth date field and a date of capture field. **Type 3:** Standardized Values requires a data file uploaded with the column names **unique** and **common** of which values from the original dataset will be mapped to one of the values in the supplied standardization dataset (*if found*).
Observe Event (Upload Standardization File Algorithm Output)	Allows the user to select a csv file for uploading their standardization mapping values.
Observe (Uploaded Standardization File)	Observes the uploaded file and will render text output of the uploaded file to help users ensure that they uploaded the correct file successfully.
Observe Event (Previous Algorithm Output IDs)	Instead of manually adding algorithm output fields, they may select from a list of existing algorithms that belong to the dataset pair. This will render a data table of all the selected algorithms output fields.
Observe Event (Use Selected Algorithms Fields)	Will take the uploaded output fields from an existing algorithm and insert them into the algorithm output fields table under the algorithm that is having output fields added.

Observe Event (Add Linkage Algorithm Output Field)	Will manually add the algorithm output field, using the field type to manage insert queries differently.
Observe Event (Drop Linkage Algorithm Output Field)	Will drop the selected algorithm output field.

Linkage Audits

Description

The linkage audits page allows the user to view saved performance measures of their algorithms, and to export them for later use.

Details

Global variables are used to keep track of the currently selected **algorithm ID** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, and a reactive value for the audit rows the user has currently selected (*to allow for select all option*).

Observe Event (Linkage Audits Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
get_audit_information()	This function performs a query that will obtain the JSON stored performance measures in the database in between the selected dates. Returning a data table.
Observe Event (Select All Audit)	Button that will select all rows in the data table generated by the **get_audit_information()** function.
Observe Event (Select None Audit)	Button that will de-select all rows in the data table generated by the **get_audit_information()** function.
Algorithm Specific Audits (RenderDataTable)	Will render the data table by taking the output from the **get_audit_information()** function.
Observe (Audit Date Change)	Will observe the two data input fields provided to the user, updating the available audit information when modified and de-selecting any selected rows.
Observe (Manual Row Selection)	Will add any manually selected rows in addition to ones selected using the **Select All** button.
Selected Algorithm Performance Measures (RenderUI)	Renders a list of definitions that can be found in the performance measures columns of the audit data table.

Observe Event (Export Selected Audits)	When clicked, the user will be prompted to select an output directory for exportation. Then, the selected rows will be binded together (*regardless of column differences*) and exported as a **csv** file.

Run Algorithm(s)

Description

The run algorithms page allows the users to run one or more algorithms at a time, choosing a variety of run-time options for what results are exported, saved, and reports are generated.

Details

Global variables are used to keep track of the currently selected **dataset pair** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, a reactive value for the algorithms that are to be ran (*to allow for select all option*).

Uploaded Linkage Output Directory (RenderText)	Initially renders the selected output directory for running linkage to reflect that it is empty.
Get Linkage Algorithms to Run (Function)	Uses the provided **dataset pair** (*IDs*) to perform a query that obtains all the currently available linkage algorithms that are runnable. Returning the result as a DT table that can have multiple rows selected.
Select Linkage Algorithms to Run (RenderDataTable)	Calls the **get_linkage_algorithms_to_run()** function, using the output to render a **Data Table UI** element.
Observe Event (Select All Run)	When clicked, all rows in the table of available algorithms to run will be selected.
Observe Event (Select None Run)	When clicked, all rows in the table of available algorithms to be run will be de-selected.
Observe (Select Linkage Algorithms to Run Rows Selected)	Will manually check for which rows are selected incase the user selects or de-selects rows after using the select all and select none buttons.
Observe Event (Linkage Output Directory)	Observes for when the user changes the linkage output directory, in which case the text box that reflects where the output will be saved is re-rendered.
Observe Event (Run Algorithm Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.

Observe Event (Linkage Report Type)	Observes which report type the user selected when running the linkage algorithms and will re-render the help text box to reflect what type of output to expect when selecting the type of report.
Observe Event (Run Linkage Button)	Uses the provided output directory, selected algorithms, and optional input values to run the main linkage. Depending on the user input, data can be saved into a **.Rds** file that is stored into the packages system files, meanwhile all other exported data files will appear in the provided output directory.

Regenerate Report

Description

The regenerate report page allows the users to use the saved **.Rdata** data located in the packages AppData to regenerate quality reports without requiring the need for re-running the algorithm and all its steps.

Details

Global variables are used to keep track of the currently selected **dataset pair** that the user selected on the **Testable Linkage Algorithms** page, a character string for which page to return to when pressing the back button, a reactive value for the algorithms that are to be ran (*to allow for select all option*).

Generate Saved Data Dataframe (Function)	Lists the saved **.Rdata** files located within the packages system files, using regular expressions to extract the **algorithm ID**, along with the **date** and **timestamp** that it was generated on. A data frame is generated containing two columns, the timestamp and the date that the report was saved.
Get Data to Regenerate (Function)	Calls the **generate_saved_data_df()** function, renames the columns to be more human readable, and returns the data as a **DT Table** object, where one or more rows may be selected.
Select Saved Data to Regenerate (RenderDataTable)	Calls the **get_data_to_regenerate()** function and uses the output to render a **DT Table UI** element.
Uploaded Regenerated Report Output Directory (RenderText)	Renders the selected output directories textbox to initially be blank.
Observe Event (Regenerated Report Output Directory)	Observes the directory that the user selected, rendering the directories textbox to reflect to the user which folder they selected.

Observe Event (Regenerate Back)	Observes for when the user presses the back button that brings the user back to the testable linkage algorithms page.
Observe Event (Regenerate Report Button)	Takes the selected rows (*algorithm data to regenerate*) and one-by-one will reload in all the. Rdata files, getting the necessary data such as linked data, algorithm summary, performances measures, etc., and will create a linkage quality report without needing to reperform the linkage process. This occurs for each of the selected saved algorithms, where a progress bar is used to show the user how far along, they are in the regeneration process.