Facilitating Search Strategy Development, Database Search & Reference Screening for Systematic Review using R

# 1. Overview

This document puts forth the functionalities for facilitating search strategy development, bibliographic database search and screening of references for systematic review using R. This also includes GUI (Graphical User Interface) to help screen the abstracts and titles of references, functionalities to assign screening effort across multiple reviewers and tools to automate the download of journal PDF articles from online databases.  
This document will be updated from time to time based on inferences, package updates and any other relevant information.

The example topic for systematic review will be ‘Screening of congenital heart disease using pulse oximetry’. Our main goals are to

* Create Boolean search string
* Search the bibliographic database using the Boolean search string
* Remove the duplicate records
* Assign the screening effort across multiple reviewers
* Familiarize with the GUI ‘Abstract Screener’
* Get the PDF of journal articles

### Requirements

* R
* RStudio

*Note: When you install an R package that needs compilation, install the binary version, if you do not have the compilation tool installed in your computer.*

# 2. Development of Boolean Search String

To create a Boolean search string, we do naïve search in bibliographic databases and systematically extract all the potential keywords from the bibliographic data we obtain. Then we will group the important keywords in to concept groups and create a Boolean search string.

### i. Get Data from Bibliographic Databases

We can perform naive search in different online medical databases and get data preferably in .bib or .txt format. Then by using the package ‘synthesisr’ we can read and merge the files in to a single data frame.

#Install the package ‘synthesisr’  
if(!require("synthesisr")) install.packages("synthesisr")

## Loading required package: synthesisr

#Load the package ‘synthesisr’  
library(synthesisr)

In this example, we use the search terms ‘congenital heart disease’ and ‘pulse oximetry’, to perform naive search in the medical databases ‘pubmed’ and ‘web of science,’ to obtain references of studies published between 2010 and 2020.

*Note: When saving results from PUBMED, choose the ‘Format’ as ‘PubMed’.*

*When saving results from WOS, select the ‘full records’ option.*

After we save the bibliographic data files in our computer, we need to import the saved files to R.

The code below demonstrates how to read and assemble bibliographic data files.

#List the bibliographic data files that we want to import  
allfiles <- list.files(here("final\_results\_10yrs\_pubTIAB\_WOSTI&AB"),full.names = TRUE)  
#Note: Here,the bibliographic data files were saved in to a file folder named 'naive\_results\_economic'.  
  
#Print the listof bibliographic data files  
print(allfiles)

## [1] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/pubmed-tele\_app\_10yrs.txt"  
## [2] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-1.txt"   
## [3] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-2.txt"   
## [4] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-3.txt"

# Citation

# allfiles\_c <- list.files("C:/Users/User/Documents/telemedicine/tele\_cost",full.names = TRUE)  
# print(allfiles\_c)  
# imported\_files\_c <- synthesisr::read\_refs(  
# filename = allfiles\_c,  
# return\_df = TRUE)  
#   
# #Find the number of records we obtained  
# nrow(imported\_files\_c)  
#   
# write.csv(imported\_files\_c, "./tele\_cost\_wos\_search.csv")  
# names(imported\_files\_c)

#Read the bibliographic data files and convert to a data frame  
imported\_files <- synthesisr::read\_refs(  
 filename = allfiles,  
 return\_df = TRUE)  
  
#Find the number of records we obtained  
nrow(imported\_files)

## [1] 2942

#Select the required variables  
imported\_files\_selected <- imported\_files[,c("title","abstract","keywords")]

### ii. Remove Duplicate Records

Now we need to remove the duplicate records. First, we can remove the duplicate articles with identical titles.

#Remove duplicate records with identical titles  
remove\_dupTitles <- synthesisr::deduplicate(  
 imported\_files\_selected,  
 match\_by = "title",  
 method = "exact"  
)  
  
#Find the number of records  
nrow(remove\_dupTitles)

## [1] 2920

Now we have a data set with deduplicated records. Still, we need to find out if there are any additional duplicate articles in our data set. For that, we can run the following script.

#We use string distance method to identify likely duplicates  
duplicates\_string <- synthesisr::find\_duplicates(  
 remove\_dupTitles$title,  
 method = "string\_osa",  
 to\_lower = TRUE,  
 rm\_punctuation = TRUE,  
 threshold = 5  
)

By running the above script, we got a data set with titles which are likely duplicates. But before we remove any articles, it is always advised to manually review the titles, to confirm that they are duplicates. Also we can extract the line numbers of the records that are likely duplicates.

#To view the records that are likely duplicates   
manual\_check <- synthesisr::review\_duplicates(remove\_dupTitles$title, duplicates\_string)  
#View(manual\_check)

On reviewing the data set of likely duplicates, if we do not find any unique titles, i.e, if we find duplicate titles for all the records, we can run the following script.

#Extracting unique references by comparing the main data set and the data set of likely duplicates  
naiveresults <- synthesisr::extract\_unique\_references(remove\_dupTitles,duplicates\_string)  
  
#Number of records after removing the duplicates  
nrow(naiveresults)

## [1] 2849

On reviewing the data set of likely duplicates, if we find any title(s) which is unique, we do not want to remove it. We can use the ‘override\_duplicates’ function to mark it as unique. Only after that, we can extract the unique references by comparing the main data set and the data set of likely duplicates.

#Mark the record with line number '22' as unique  
#new\_duplicates <- synthesisr::override\_duplicates(duplicates\_string,22)  
  
#Extracting unique references by comparing the main data set and the new data set of duplicates  
#naiveresults <- synthesisr::extract\_unique\_references(remove\_dupTitles,new\_duplicates)

*Note: In our data set of likely duplicates, we could not find any titles which are unique. So we are skipping the above code lines. But in your data set, if you find any records which are unique, you must run the above script so that the unique references will not be deleted.*

### iii. Identify Potential Keywords

Now we need to identify the potential keywords from deduplicated records. For that, we can use the functions from the package ‘litsearchr’.

#Install the package 'litsearchr'  
#devtools::install\_github("elizagrames/litsearchr")  
  
#Load the package 'litsearchr'  
library(litsearchr)

The ‘extract\_terms’ function of the package ‘litsearchr’ will systematically extract all potential keywords from the article titles, abstracts, or other fields that are passed to the function as text.

#Extract all potential keywords from article titles and abstracts  
rakedkeywords <-  
 litsearchr::extract\_terms(  
 text = paste(naiveresults$title, naiveresults$abstract),  
 method = "fakerake",  
 min\_freq = 2,  
 ngrams = TRUE,  
 min\_n = 2,  
 language = "English"  
 )  
  
#Extract the author/database-tagged keywords.  
taggedkeywords <- litsearchr::extract\_terms(keywords = naiveresults$keywords, ngrams = TRUE, min\_n=1,method="tagged",language = "English")

### iv. Build the Keyword Co-Occurrence Network

We run a series of functions to create a co-occurrence network and then identify the potential keywords.

#Create keyword co-occurance network  
all\_keywords <- unique(append(taggedkeywords, rakedkeywords))  
  
naive\_dfm <-  
 litsearchr::create\_dfm(  
 elements = paste(naiveresults$title, naiveresults$abstract),  
 features = all\_keywords  
 )  
naivegraph <-  
 litsearchr::create\_network(  
 search\_dfm = as.matrix(naive\_dfm),  
 min\_studies = 2,  
 min\_occ = 2  
 )

### v. Identify Change Points in Level of Importance of Keywords

The keyword co-occurrence network can next be quantitatively assessed to detect a cutoff point for changes in the level of importance of a particular keyword to the concept. This will help in making an efficient but comprehensive search by removing terms that are not central to a field of study.

cutoff <- litsearchr::find\_cutoff(naivegraph, method = "cumulative", percent = .80,imp\_method = "strength")  
  
reducedgraph <- litsearchr::reduce\_graph(naivegraph, cutoff\_strength = cutoff[1])  
  
searchterms <- litsearchr::get\_keywords(reducedgraph)  
  
#Display the first 20 keywords  
head(searchterms,20)

## [1] "-forward" "abortion"   
## [3] "acceptability" "acceptance"   
## [5] "access" "access to care"   
## [7] "access to health care" "accessibility"   
## [9] "activities of daily living" "acute ischemic stroke"   
## [11] "acute myocardial infarction" "acute stroke"   
## [13] "acute stroke care" "addiction"   
## [15] "adherence" "administration"   
## [17] "adolescent" "adolescents"   
## [19] "adult" "africa"

#saving the search terms to a .csv file  
write.csv(searchterms, "./tele\_pol\_impl\_terms.csv")  
  
#Display all the keywords  
searchterms

### vi. Group Terms in to Similar Concept Groups

Now that the important keywords for the search have been identified, they can be grouped into blocks to build the search strategy. For Boolean searches, terms are grouped into similar concept groups where they can be combined with “OR” statements and the separate blocks combined with “AND” statements.

It is recommended to save the search terms to a .csv file, adding a new column called “group”, and entering the group names in it, then reading in the .csv file. Although this can be done in R, adding tags to 400+ suggested search terms is generally quicker in a .csv file.

In our example, all keywords that relate to congenital heart disease would be in a similar concept group (e.g., “Atrial septal defect”, “Pulmonary Atresia” etc.) while terms related to pulse oximetry (e.g. “oxygen saturation”, “preductal saturation”, etc.) would be in their own concept group. Terms that fit multiple concept groups can be added to both without changing the logic of the Boolean connections.

#Import the csv after making the necessary changes  
grouped\_terms <- read.csv("./tele\_economic\_terms\_screened.csv")  
  
#Extract the ‘congenital HD’ terms from the csv  
telemedicine\_terms <- grouped\_terms$x[which(stringr::str\_detect(grouped\_terms$group, "Telemedicine"))]  
  
#Join together a list of manually generated ‘congenital HD’ terms with the ones from the csv  
#Note: Manually generated terms are the terms which we think, are important keywords; but are not present in the .csv file  
# telemedicine <- unique(append(c("tele-ophthalmology","tele-radiology","tele-mental health","tele-pathology","tele-dentistry","tele-cardiology","tele-oncology","tele-rehabilitation"), as.character(telemedicine\_terms)))  
  
telemedicine <- unique(as.character(telemedicine\_terms))  
  
#Extract the ‘pulse oximetry’ terms from the csv  
economic\_terms <- grouped\_terms$x[which(stringr::str\_detect(grouped\_terms$group, "Economic"))]  
  
#Join together a list of manually generated ‘pulse oximetry’ terms with the ones from the csv. If there are no manually generated terms, we can simply use the terms from the csv.  
# application <- unique(append(c("use\*","advantage","usage","benefit"),as.character(application\_terms)))  
  
economic <- unique(as.character(economic\_terms))  
  
#Note: Repeat this for all concept groups  
  
#Then merge them into a list  
mysearchterms <- list(telemedicine,economic)

### vii. Write Boolean Searches

Once keywords are grouped into concept groups in a list, the ‘write\_search’ function can be used to write Boolean searches in multiple languages. The script below demonstrates writing a search in English using the search terms.

#Generate Boolean search string  
boolean\_string <- litsearchr::write\_search(  
 groupdata = mysearchterms,  
 languages = "English",  
 exactphrase = FALSE,  
 stemming = FALSE,  
 closure = "none",  
 writesearch = FALSE,  
 verbose = TRUE  
)

## [1] "English is written"

#View the boolean search string generated  
boolean\_string

## [1] "((telemedicine OR telerehabilitation OR mhealth OR (mobile health) OR teledermatology OR telemonitoring OR teleophthalmology OR telepsychiatry OR ehealth OR (electronic health) OR e-health OR m-health OR (digital health) OR telestroke OR teleconsultation OR e-consult OR (health technology assessment)) AND ((economic burden) OR cost OR (economic benefits) OR (economic evaluation) OR qaly OR (economic analysis) OR (economic impact) OR (economic outcomes) OR (health economic)))"

# 3. Database Search using the Boolean Search String

### i. Import Data from Bibliographic Database using Boolean Search String

Now using the Boolean search string developed, we can do the final search in the bibliographic databases and save the result data files in our computer. Then we need to read and merge all the files in to a single data frame and then remove the duplicate records.

*Note: When saving results from PUBMED, choose the ‘Format’ as ‘PubMed’.*

*When saving results from WOS, select the ‘full records’ option in .txt format.*

#List the bibliographic data files after the final search  
allfiles\_final <- list.files(here("final\_results\_10yrs\_pubTIAB\_WOSTI&AB"),full.names = TRUE)  
  
#Print the list of bibliographic data files  
print(allfiles\_final)

## [1] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/pubmed-tele\_app\_10yrs.txt"  
## [2] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-1.txt"   
## [3] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-2.txt"   
## [4] "C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija/final\_results\_10yrs\_pubTIAB\_WOSTI&AB/wos-tele-app-10yrs-3.txt"

#Read the bibliographic data files and convert to a data frame  
final\_files <- synthesisr::read\_refs(  
 filename = allfiles\_final,  
 return\_df = TRUE)  
  
#Find the number of records we obtained  
nrow(final\_files)

## [1] 2942

#### If only WOS files(.txt) are to be screened, run the following script.

final\_files\_selected <- final\_files[,c("author\_full","title","source","year","doi","abstract","language","keywords","volume")]  
  
final\_files\_selected <- final\_files\_selected %>% dplyr::rename(journal = source)

#### If only pubmed files(.txt) are to be screened, run the following script.

#Select the required variables  
filtered\_final\_files <- final\_files[,c("author\_full","title","journal","abstract","language","keywords","date\_published","article\_id","volume")]  
  
#Extract the year published from pubmed records  
pub\_year\_final <- gsub( " .\*$", "", filtered\_final\_files$date\_published)  
filtered\_final\_files <-filtered\_final\_files %>%   
 dplyr::mutate(year= pub\_year\_final)  
  
#Extract doi from pubmed records  
before\_doi\_final<- sub("\\[doi].\*", "", filtered\_final\_files$article\_id)  
after\_and\_final <- sub(".\*and", "", before\_doi\_final)  
after\_and\_final <- trimws(after\_and\_final, which = c("left"))  
filtered\_final\_files <-filtered\_final\_files %>%   
 dplyr::mutate(doi= after\_and\_final)  
  
#Convert column fields with no proper doi to NA  
is.na(filtered\_final\_files$doi) <- !(startsWith(filtered\_final\_files$doi, "10."))  
  
#Select the required variables  
final\_files\_selected <- filtered\_final\_files[,c("author\_full","title","journal","year","doi","abstract","language","keywords","volume")]

#### If both pubmed(.txt) and wos(.txt) files are to be screened together, run the following script

#Extract required variables  
filtered\_final\_files <- final\_files[,c("author\_full","title","journal","source","year","doi","abstract","language","keywords","date\_published","article\_id","volume","filename")]  
  
#Fill the empty column fields in 'journal'  
filtered\_final\_files$journal <- ifelse(is.na(filtered\_final\_files$journal), filtered\_final\_files$source, filtered\_final\_files$journal)  
  
#Extract the year published from pubmed records  
pub\_year\_final <- gsub( " .\*$", "", filtered\_final\_files$date\_published)  
filtered\_final\_files$year <- ifelse(is.na(filtered\_final\_files$year), pub\_year\_final, filtered\_final\_files$year)  
  
#Extract doi from pubmed records  
before\_doi\_final<- sub("\\[doi].\*", "", filtered\_final\_files$article\_id)  
after\_and\_final <- sub(".\*and", "", before\_doi\_final)  
after\_and\_final <- trimws(after\_and\_final, which = c("left"))  
filtered\_final\_files$doi <- ifelse(is.na(filtered\_final\_files$doi), after\_and\_final, filtered\_final\_files$doi)  
  
#Convert column fields with no proper doi to NA  
is.na(filtered\_final\_files$doi) <- !(startsWith(filtered\_final\_files$doi, "10."))  
  
#Select variables after data cleaning  
final\_files\_selected <- filtered\_final\_files[,c("author\_full","title","journal","year","doi","abstract","language","keywords","volume")]  
nrow(final\_files\_selected)

## [1] 2942

Now we need to remove the duplicate records. First, we can remove the duplicate articles with identical titles.

#Remove duplicate records with identical titles  
remove\_dup\_final <- synthesisr::deduplicate(  
 final\_files\_selected,  
 match\_by = "title",  
 method = "exact"  
)  
#Find number of records  
nrow(remove\_dup\_final)

## [1] 2920

Now we got a data set with deduplicated records. Still, we need to find out if there are any additional duplicate articles in our data set. For that, we can run the following script.

#We use string distance method to identify likely duplicates  
duplicates\_string\_final <- synthesisr::find\_duplicates(  
 remove\_dup\_final$title,  
 method = "string\_osa",  
 to\_lower = TRUE,  
 rm\_punctuation = TRUE,  
 threshold = 5  
)

By running the above script, we got a data set with titles which are likely duplicates. But before we remove any articles, it is always better to manually review the titles, to confirm that they are duplicates. Also we can extract the line numbers of the records that are likely duplicates.

#To view the records that are likely duplicates   
manual\_check\_final <- synthesisr::review\_duplicates(remove\_dup\_final$title, duplicates\_string\_final)  
#View(manual\_check\_final)

On reviewing the data set of likely duplicates, if we do not find any unique titles, i.e, if we find duplicate titles for all the records, we can run the following script.

#Extracting unique references by comparing the main data set and the data set of likely duplicates  
string\_res <- synthesisr::extract\_unique\_references(remove\_dup\_final,duplicates\_string\_final)  
  
#Number of records after removing the duplicates  
nrow(string\_res)

## [1] 2849

On reviewing the data set of likely duplicates, if we find any title(s) which is unique, we do not want to remove it. We can use the ‘override\_duplicates’ function to mark it as unique. Only after that, we can extract the unique references by comparing the main data set and the data set of likely duplicates.

#Mark the record with line number '762' as unique  
#new\_duplicates\_final <- synthesisr::override\_duplicates(duplicates\_string\_final,762)  
  
#Extracting unique references by comparing the main data set and the new data set of likely duplicates  
#string\_res <- synthesisr::extract\_unique\_references(remove\_dup\_final,new\_duplicates\_final)

*Note: In our data set of likely duplicates, we could not find any titles which are unique. Hence we are skipping the above code lines. But in your data set, if you find any records which are unique, you must run the above script so that unique references will not be deleted.*

Now we have a data set with unique references to be screened.

#The bibliographic variables in this dataset can be displayed by running the following script  
names(string\_res)

## [1] "author\_full" "title" "journal" "year" "doi"   
## [6] "abstract" "language" "keywords" "volume" "n\_duplicates"

#Display the titles  
titles<-string\_res["title"]

# 4. Screening of References

### i. Delegate Reference Screening Effort to a Team

Now our goal is to distribute the screening effort to two or more reviewers. For that, we can use functions from the package ‘metagear’.

The ‘metagear’ package can be installed by running the following R script

#Install the packages ‘BiocManager’ & 'EBImage'  
#install.packages("BiocManager")  
#BiocManager::install("EBImage")

#Load the package ‘metagear’  
library(metagear)  
#Note: The first loading of metagear using library(metagear) will trigger the download of the gWidgets package and associated toolkits needed to build GUI interfaces. A small window will also prompt you to download GTK+ asking “Need GTK+ ?”. From the listed options answer: “Install GTK+” and click “OK”. Once installed, these will not be downloaded again.

In our example, we are distributing the screening efforts to three reviewers: “ReviewerA”, “ReviewerB” and “ReviewerC”. Each reviewer will screen a separate subset of these references.

First, we need to initialize this data set for screening tasks.

#Initialize the data set  
ref <- effort\_initialize(string\_res)  
  
#Display the new columns added by ‘effort\_initialize’ function  
names(ref)

## [1] "STUDY\_ID" "REVIEWERS" "INCLUDE" "author\_full" "title"   
## [6] "journal" "year" "doi" "abstract" "language"   
## [11] "keywords" "volume" "n\_duplicates"

Note that the effort\_initialize() function added three new columns: “STUDY\_ID” which is a unique number for each reference, “REVIEWERS” an empty column with NAs that will be later populated with our reviewer names, and the “INCLUDE” column, which will later contain the screening efforts by the reviewers. Screening efforts are essentially how individual study references get coded for inclusion in the synthesis project; currently the “INCLUDE” column has each reference coded as “not vetted”, indicating that each reference has yet to be screened.

# Change the format of title, abstract & journal in the dataset  
ref$title <- stringr::str\_to\_sentence(ref$title)  
ref$abstract <- stringr::str\_to\_sentence(ref$abstract)  
ref$journal <- stringr::str\_to\_sentence(ref$journal)

Next task is to delegate screening efforts to our three reviewers; ReviewerA, ReviewerB and ReviewerC. These references will be randomly distributed to each reviewer. For that, we can use effort\_distribute() function.

By setting the ‘save\_split’ parameter to TRUE, the reference subset will be saved in separate files. These can be given to corresponding reviewers to start their work.

The set.seed() function sets the starting number used to generate a sequence of random numbers – it ensures that you get the same result if you start with that same seed each time you run the same process.

#Randomly distribute screening effort to three reviewers and saving the screening tasks to a separate file  
team<- c("Ashis")  
set.seed(123);  
unscreened <- metagear::effort\_distribute(ref,reviewers = team, save\_split= TRUE)

## File already existed, renamed: effort\_Ashis2.csv

## 1 files saved in: C:/Users/Arun/Dropbox/My PC (DESKTOP-KFF959O)/Documents/telemedicine\_to\_adrija

#3 files will be saved in the file folder  
  
#Display screening tasks  
unscreened[c("STUDY\_ID", "REVIEWERS")]  
  
#View the files created  
list.files(pattern = "effort")  
#Note: It will list all the files with names starting with 'effort' in that particular folder

We can see three separate effort\_\*.csv files created with references to be screened. These can be passed on to the corresponding team member so that they can begin the screening of references.

In case, we do not want to distribute the screening task equally among all the team members, we also have option to distribute the task unevenly, such as below where ReviewerA will take on 60% of the screening effort and the other two reviewers 20% each:

set.seed(123);  
unscreened <- metagear::effort\_distribute(ref, reviewers = team, effort = c(60,20,20))  
  
unscreened[c("STUDY\_ID", "REVIEWERS")]

The effort can also be redistributed with the effort\_redistribute() function. Here we are assigning 10% of ReviewerA’s work to a new team member ReviewerD.

#Redistributing the work  
#Moving 10% of ReviewerA's work to ReviewerD  
set.seed(123);  
ref\_redistribute <- metagear::effort\_redistribute(unscreened,reviewer = "ReviewerA", remove\_effort = 10, reviewers =c("ReviewerA","ReviewerD"))  
  
#Team members loosing and picking up work  
ref\_redistribute [c("STUDY\_ID", "REVIEWERS")]

### ii. Abstract Screener

The package ‘metagear’ offers a simple abstract screener to quickly screen the abstracts and titles of multiple references.

Given below is the script to help initialize the GUI in R:

# Custom abstract screener function with year and journal added  
source('abstract\_screener\_custom\_function.R')   
  
# Illustration of the custom widget with Year and Journal added  
abstract\_screener\_custom("effort\_Ashis.csv",  
 aReviewer = "Ashis")

The GUI itself will appear as a single window with the first title/abstract listed in the .csv file. If abstracts have already been screened, it will begin at the nearest reference labeled as “not vetted”. The YES, MAYBE, NO buttons, which also have shortcuts ALT-Y and ALT-N, are used to code the inclusion/exclusion of the reference. Once clicked/coded the next reference is loaded. References should be coded as “YES” or “NO” for inclusion, but can also be coded as “MAYBE” if bibliographic information is missing or there is inadequate information to make a proper assessment of the study. The SAVE button is used to save the coding progress of screening tasks. It will save coding progress directly to the loaded .csv file. The SEARCH WEB button opens the default browser and searches Google with the title of the reference. Closing the GUI without saving will result in the loss of screening efforts relative to last save.

Once the screening is complete, we have an option to merge the three files with the completed screening efforts using the effort\_merge() function, as well as summarize the outcome of screening tasks using the effort\_summary() function.

#Merge the screened studies, effort\_ReviewerA.csv, effort\_ReviewerB.csv and effort\_ReviewerC.csv  
#WARNING: will merge all files named "effort\_\*" in directory  
screened <- effort\_merge()  
screened[c("STUDY\_ID", "REVIEWERS", "INCLUDE")]  
theSummary <- effort\_summary(screened)

The summary of screening tasks describes which references are appropriate, while also outlining which need to be re-assessed. The team should discuss these references and decide if they are appropriate for inclusion or track down any additional/missing information needed to make proper assessment of their inclusion.

# 5. Download the PDF of Journal Articles

Once the references have been screened, we can download the PDF articles of selected references by using the ‘metagear’ package. First, we can extract the selected references from the screened .csv file by using the following script.

#Read the .csv file with screened references  
vetted <- read.csv("./effort\_Ashis\_tele\_economic\_evaluation\_screened.csv")  
  
#Select only the selected references  
selected\_ref <- subset(vetted,INCLUDE=='YES')  
  
selectedRefs\_download <- selected\_ref[,c("author\_full","title","journal","year","doi","abstract","language","keywords","volume")]

Now we have a .csv file with only the selected references. By using this file, we can download the PDF articles by running the following R script.

*Note that the download success of these PDFs is entirely conditional on the journal subscription coverage of the host institution running metagear. Also note that metagear only supports the download of a PDF article if the DOI (digital object identifier) is available for that article.*

#Extract last name of author  
last\_name<- sub("\\,.\*", "", selectedRefs\_download$author\_full)  
  
#Add extracted last name to dataset  
selectedRefs\_download <-selectedRefs\_download %>%   
 mutate(last\_name= last\_name)  
  
#Combine the title,lastname & year  
selectedRefs\_download$title\_download <- paste(selectedRefs\_download$title,selectedRefs\_download$last\_name,selectedRefs\_download$year,sep=";")  
  
#Create a directory where the downloaded PDFs will be saved  
dir.create("PDF\_downloads\_economic\_jio")  
  
#Fetch the PDFs  
collectionOutcomes <- metagear::PDFs\_collect(selectedRefs\_download, DOIcolumn = "doi", FileNamecolumn = "title\_download", directory = "PDF\_downloads\_economic\_jio",quiet = TRUE)  
#Note: 'Column name of DOIs in your data set must be assigned to the variable 'DOIcolumn'  
  
#Check the download success  
table(collectionOutcomes$downloadOutcomes)

# 6. Load References in Zotero

We can load the selected references in Zotero and use all the functionalities of Zotero to manage our selected references. We can use the functions of the package ‘zoteror’ to access the zotero API and then load our references in Zotero.

#Install & load the package 'zoteror'  
if(!require("remotes")) install.packages("remotes")

## Loading required package: remotes

#remotes::install\_github(repo = "giocomai/zoteror")  
  
#Load the package 'zoteror'  
library(zoteror)

To create a zotero API, we need to log in to zotero website, go to the (Feeds/API page in Settings)[<https://www.zotero.org/settings/keys>], and create an API from there. The API created will be set as ‘credentials’ in our R script.

#Store credentials  
#Note: Instead of inputting user and credentials each time a function is run, it is easier to set them with 'ZotSetOptions' at the beginning of every session.  
zot\_set\_options(user = 6574132, credentials = "RVpzjCu9nqwwTBNrTt8aRQlP")  
  
#Note: user will be your user ID & credentials will be your API Key created. To execute the above script, you need to provide your user ID and API key created.

Now we can create a new collection in Zotero by the given name, and outputs its key. If a collection with the same name already exists, it does not create a new one, but rather outputs the key of the pre-existing collection. The function zot\_create\_collection() requires an API key with write access.

#Load the package 'tibble'  
library(tibble)  
  
#Create a collection   
key <- zot\_create\_collection(collection\_name = "References")  
#Note: key\_CHD contains the key of the collection created

Now on opening the Zotero, we can see a new folder created in the given name. At present, the folder will be empty. Now we can load our references in that folder by running the following R script.

#Read the .csv file with screened references  
vetted <- read.csv("./effort\_ReviewerA\_screened.csv")  
  
#Select only the selected references to be loaded in Zotero  
zotero\_items <- subset(vetted,INCLUDE=='YES')  
  
  
#Transform our reference data frame in to a format that fully mirrors Zotero's data structure  
library(tidyverse)  
zotero\_items\_formatted <-   
 zotero\_items %>%   
 mutate(itemType= 'journalArticle',title=title,abstractNote=abstract,  
 creators = zot\_convert\_creators\_to\_df\_list(creator = author\_full,creator\_type = "author",separator\_multiple\_authors = " and ",separator\_surname\_name = ", "),   
 publicationTitle = journal, volume = volume, language = language, date = year, DOI=doi)

#Create a dataframe with required fileds to be loaded in Zotero  
df = zotero\_items\_formatted[c("itemType","title","abstractNote","creators",  
 "publicationTitle","volume","language","date","DOI")]  
  
#Upload the items in Zotero  
zot\_create\_items(item\_df = df, collection = "References")

Now, on checking the Zotero, we can find the items have been loaded. We can right-click on the items and select ‘Find available PDF’ to get the PDF of the articles. If PDF is not available for download, there is an option to view the articles online.

# 7. References

## i. CRAN.R - Packages Documentation

1. Metagear Package - Documentation

URL : <https://cran.r-project.org/web/packages/metagear/metagear.pdf>

1. Synthesisr Package – Documentation

URL : <https://cran.r-project.org/web/packages/synthesisr/synthesisr.pdf>

## ii. GitHub Repo

1. Litsearchr Package

URL : <https://github.com/elizagrames/litsearchr>

1. Zoteror Package

URL: <https://github.com/giocomai/zoteror>