Journal of Statistical Software
June 2012, Volume 49, Issue 5. *http://www.jstatsoft.org/*

Closing the Gap between Methodologists and End-Users: R as a Computational Back-End

Byron C. Wallace
Tufts Medical Center

Issa J. Dahabreh
Tufts Medical Center

Thomas A. Trikalinos
Tufts Medical Center

Joseph Lau
Tufts Medical Center

Paul Trow
Tufts Medical Center

Christopher H. Schmid
Tufts Medical Center

Abstract

The R environment provides a natural platform for developing new statistical methods due to the mathematical expressiveness of the language, the large number of existing libraries, and the active developer community. One drawback to R, however, is the learning curve; programming is a deterrent to non-technical users, who typically prefer graphical user interfaces (GUIs) to command line environments. Thus, while statisticians develop new methods in R, practitioners are often behind in terms of the statistical techniques they use as they rely on GUI applications. Meta-analysis is an instructive example; cutting-edge meta-analysis methods are often ignored by the overwhelming majority of practitioners, in part because they have no easy way of applying them. This paper proposes a strategy to close the gap between the statistical state-of-the-science and what is applied in practice. We present open-source meta-analysis software that uses R as the underlying statistical engine, and Python for the GUI. We present a framework that allows methodologists to implement new methods in R that are then automatically integrated into the GUI for use by end-users, so long as the programmer conforms to our interface. Such an approach allows an intuitive interface for non-technical users while leveraging the latest advanced statistical methods implemented by methodologists.

Keywords: meta-analysis, software, open source, Python, R.

1. Introduction

In statistically-driven sciences there is often a disparity between cutting edge methodologies and the techniques used by practitioners in their daily work. This disparity is due in part to a dearth of technical know-how. Methodologists (usually statisticians) tend to implement advanced methods in statistical programming languages such as Stata and SAS, and perhaps most commonly now, in R (R Development Core Team 2012). Non-technical researchers, meanwhile, are often deterred by programming and rely on available graphical user interface (GUI) driven tools. Such specialized software is generally easy to use, but usually lags with respect to the statistical methods available. The result is that practitioners continue to use inappropriate or suboptimal methods due to their being restricted to what is made available via GUIs. The problem is compounded by the fact that statistical programmers generally do not implement custom GUIs for new methods they develop.

The example of meta-analysis, described at length in the following section, motivates our work. Most meta-analysts are non-technical clinical researchers without experience in R or any other statistical programming language. They therefore tend to rely on available (often commercial) GUI software, e.g., **Comprehensive Meta-Analysis** (**CMA**, Borenstein, Rothstein, and Cohen 2005). Such programs are user-friendly, but it is difficult for the maintainers to keep up-to-date with the latest methodological developments. For example, despite their user-friendly interfaces, no existing GUI-driven specialized meta-analysis software that we are aware of accomodates multivariate datasets with multiple correlated outcomes, or longitudinal data on multiple treatments (in a network of treatments). Furthermore, most analytic choices are presented to the user as defaults with few, if any, options for more advanced analyses. Finally, graphs are often of low quality and the software architecture cannot easily accomodate new graphing capabilities.

Indeed, we encountered the above difficulties firsthand when implementing and maintaining our previous version of meta-analysis software, **Meta-Analyst** (Wallace, Schmid, Lau, and Trikalinos 2009). The program was implemented in C#, atop the .NET platform. Although the .NET platform made it easy to develop a user-friendly interface (spreadsheet-driven, etc.), it practically restricted us to the Windows platform.[1] More importantly, however, the language is not ideal for statistical programming; we found ourselves porting existing code from R to C#, an error-prone and inefficient process.

In this work, we propose a novel strategy for making advanced methods available to non-technical end-users. We apply this strategy to meta-analysis as an example, but we believe it to be relatively general. We use a software architecture where an 'agnostic' front-end GUI (in our case written in Python) interfaces with a calculation and graphics engine (in R) in the back-end, using relatively simple standards. The GUI is agnostic in that it can render inputs to and results from arbitrary routines implemented in the calculations engine, so long as they conform to the predefined standards. The architecture lends itself to a 'two-tiered' distribution of statistical methods; advanced/technical users may opt to use the underlying R package(s) directly, ignoring the GUI, while non-technical users may only use the GUI, unaware even that R is the back-end.

2. Meta-analysis

Meta-analysis is the quantitative synthesis of information from independent sources (studies). It is typically used to answer questions that individual studies are underpowered or

[1]Technically, we may have been able to run the program on top of alternative .NET implementations, e.g., Mono, however such implementations are not yet mature.

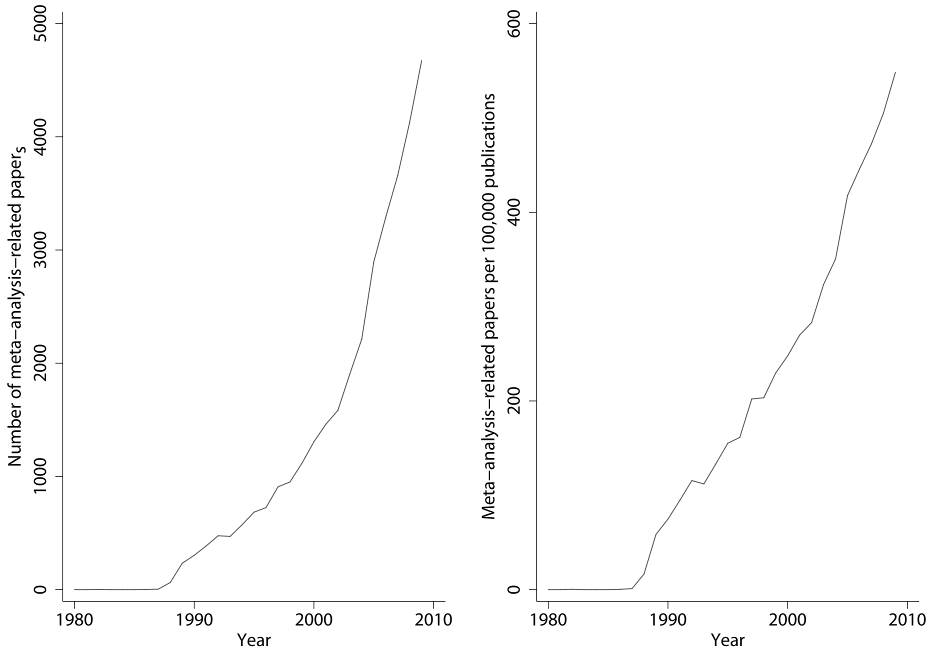


Figure 1: Growth of the meta-analysis literature in the biomedical sciences. Based on a MEDLINE search (using the keywords: 'meta-analysis[Publication Type]' and 'meta-analysis as topic[MeSH Terms]'), we obtained estimates of the crude number (left panel), and the proportion (per 100,000 publications, right panel) of published papers related to meta-analysis, respectively, over the last 3 decades. The left panel represents the growth in the absolute number of meta-analysis-related publications whereas the right pannel demonstrates the growth of the meta-analysis field normalized to the overall growth of the literature.

not designed to address. Meta-analysis has a long tradition in many scientific disciplines, including the natural sciences, such as physics, astronomy (Airy 1875) and ecology (Osenberg, Sarnelle, and Goldberg 1999), as well as medicine (Simpson and Pearson 1904; Fisher 1925), epidemiology (Stroup *et al.* 2000), psychology (Glass 1976), and education (Kulik and Kulik 1989). As the number of primary studies in all scientific fields has increased, so too has the need to synthesize this evidence using meta-analytic methods. For example, Figure 1 shows the exponential growth of meta-analyses and related publications in the medical literature over the last 3 decades. At the same time methodological advances have made it possible to the synthesize complex data, including multivariate data on multiple correlated outcomes, data assessed over successive time points (longitudinal data), or data on multiple treatments (networks of treatments).

2.1. Existing software for meta-analysis

The tasks involved in evidence synthesis are complex, and in order to accomodate this complexity, meta-analysis software should ideally:

- Allow data entry for complex datasets, including multiple outcomes reported in each study, simultaneous comparisons of multiple interventions across studies (networks of contrasts) and reporting of results over multiple timepoints (longitudinal datasets).

- Facilitate back-calculation of sufficient statistics for meta-analysis based on the the data reported by each study. For example, the software could back-calculate a variance (a statistic that is often necessary for meta-analysis) from the corresponding p-value or confidence interval, or other information. Automating such calculations can reduce the reviewers' work burden and minimize data extraction errors (Gotzsche, Hrobjartsson, Maric, and Tendal 2007).
- Include most commonly used meta-analysis methods and, importantly, allow the addition of new statistical techniques as they become available.
- Implement Bayesian meta-analysis methods through Markov chain Monte Carlo simulations.
- Generate publication quality graphs for data presentation and exploration and allow the rendering of new graph types based on the particular requirements of novel statistical methods.
- Be open-source to ensure availability in resource-limited settings and to allow community involvement in development and testing.
- Combine the above listed technical requirements with ease of use, ideally through a GUI.

Currently, meta-analysis methods are implemented in many statistical packages, including Stata (Sterne 2009), SPSS (Lipsey and Wilson 2001), SAS (Arthur, Bennett, Huffcutt, Arthur, and Bennett 2001) and R (Schwarzer 2012; Lumley 2009; Viechtbauer 2010), in addition to Microsoft **Excel** plug-ins, such as **MIX** (Bax, Yu, Ikeda, Tsuruta, and Moons 2006) and **MetaEasy** (Kontopantelis and Reeves 2009). Stand-alone software for general meta-analysis include **Comprehensive Meta-Analysis** (Borenstein *et al.* 2005), **MetaWin** (Rosenberg, Adams, and Gurevitch 1997), **Meta-Analyst** (Wallace *et al.* 2009), and the Cochrane Collaboration's **RevMan** (Clarke and Oxman 2000). In addition, there is dedicated software for specialized types of meta-analysis, such as meta-analysis of diagnostic test data; examples are **Meta-Test** (Lau 1997) and **Meta-Disc** (Zamora, Abraira, Muriel, Khan, and Coomarasamy 2006). We have provided a more exhaustive overview elsewhere (Wallace *et al.* 2009).

The methodologies that are implemented in software end up being widely used, because they are readily available. A striking example is the almost universal use of the DerSimonian-Laird method for random effects univariate meta-analysis (DerSimonian and Laird 1986) (the default or only method in the aforementioned programs), irrespective of good data that alternative random effect methods have better statistical properties (Turner, Omar, Yang, Goldstein, and Thompson 2000; Viechtbauer 2005).

In the following sections we describe the proposed software architecture by virtue of an applied example, namely our new meta-analysis program, which we call **OpenMeta-Analyst**.

3. Why R and Python?

We will briefly explain our reasons for electing to use the R and Python languages for **OpenMeta-Analyst**. R has become the lingua franca amongst methodologists and statisticians. We argue that Python is in many ways ideal for implementing a GUI; it is cross-platform, has a large, open-source community and the **rpy** library (Moriera and Warnes 2004) facilitates easy communication with R. Moreover, there are many mature, rich GUI toolkits available in Python. We feel that general purpose programming languages provide a better framework for writing GUIs compared to specialized statistical languages such as R.

There are many major benefits to using R as a statistical engine. First, doing so allows one to exploit the wealth of methods already written in R. By the same token, it allows us to incorporate new methods that are written with relatively little effort (i.e., we need only 'wrap' them so as to conform to our interface, as explicated in Section 4.2). Second, R is both cross-platform and open-source, both of which are desirable traits in any scientific software.

Our approach is similar in spirit to the R Commander package (**Rcmdr**), which provides an "easy-to-use, cross-platform, extensible GUI" with enough functionality to span a basic statistics course (Fox 2005). Like **Rcmdr**, we hope to provide an inuitive GUI that exploits the power of the R environment. However, rather than creating a general statistics program, we aimed to develop an intuitive, easy-to-use program for conducting meta-analyses that leverages the existing R code for this task. Moreover, we wanted this architecture to be extensive, so that new statistical routines can be developed by methodologists and plugged into the GUI with minimal effort.

For the GUI, we elected to use Python, rather than a GUI toolkit native to R such as the **tcltk** package, as the author of **Rcmdr** did. We did so for a few reasons. First, we were able to use the **PyQT** library (River Bank Computing 2012), a mature GUI toolkit written in Python. We feel that **PyQT** provides a richer set of GUI widgets than does **tcltk**. Second, Python is a cross-platform, open-source, object-oriented programming language, which we feel is more conducive to writing GUI code than is R. In any case, we note that had we opted to use **tcltk**, our approach to communicating between the GUI and underlying methods would remain as it is outlined in the following section.

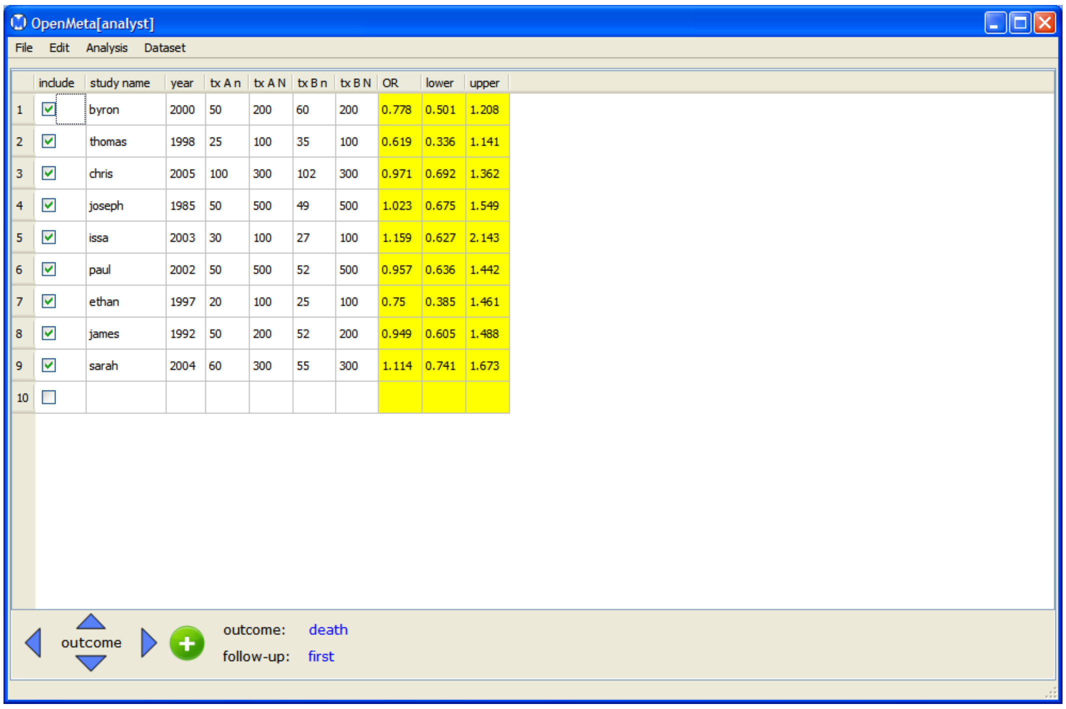
The GUI we built with **PyQT** is shown in Figure 2. Because **PyQT** is a modern, mature graphics package, we were able to include such features as a full-fledged spreadsheet component, unlimited undo/redo functionality, and other modern interface features with relative ease. Moreover, **QT** widgets emulate the native look of the target platforms (e.g., a button will appear as a standard Windows button on Windows, and a standard OS X button on OS X), which is aesthetically pleasing.

4. Communication between statistical methods and the GUI

We will now describe the interface between the Python and R elements that comprise **OpenMeta-Analyst**. The key idea is that by exploiting assumptions about naming conventions and argument passing, methods written in R can be plugged in seamlessly to the GUI, meaning that they will be available to end-users via the GUI without the R coder having to write any GUI code. We will first outline our general framework for accomplishing this, and then walk through a concrete example.

4.1. The general framework

The main tenet of the design of **OpenMeta-Analyst** is that the user interface must be entirely divorced from the underlying statistical routines. In particular, the Python interface is to



	include	study name	year	tx A n	tx A N	tx B n	tx B N	OR	lower	upper
1	✓	byron	2000	50	200	60	200	0.778	0.501	1.208
2	✓	thomas	1998	25	100	35	100	0.619	0.336	1.141
3	✓	chris	2005	100	300	102	300	0.971	0.692	1.362
4	✓	joseph	1985	50	500	49	500	1.023	0.675	1.549
5	✓	issa	2003	30	100	27	100	1.159	0.627	2.143
6	✓	paul	2002	50	500	52	500	0.957	0.636	1.442
7	✓	ethan	1997	20	100	25	100	0.75	0.385	1.461
8	✓	james	1992	50	200	52	200	0.949	0.605	1.488
9	✓	sarah	2004	60	300	55	300	1.114	0.741	1.673
10	☐									

Figure 2: A screenshot of the GUI we have developed using the **PyQT** library; this, in turn, interfaces with R to perform statistical analyses.

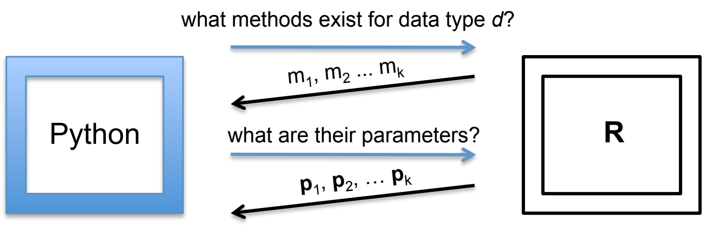

what methods exist for data type *d*?
$m_1, m_2 \ldots m_k$
what are their parameters?
$p_1, p_2, \ldots p_k$
Python
R

Figure 3: Python requests a list of available methods for a particular data type from R. The parameter names and types required by these methods are also requested. Knowing the type of parameter allows an interface to be generated, on the fly.

know only that it is calling an abstract *type* of method (e.g., a meta-analysis method), and not the concrete method that it is actually invoking. We accomplish this through the dual strategy of assuming certain naming standards and exploiting R's introspective capabilities (i.e., ability to dynamically examine the current run-time environment).

The meta-analysis methods themselves are contained in our R package **openmetar**. This is a stand-alone package that in the future may be useful on its own for users interested in bypassing the GUI. At present, the package serves as an *adapter* that 'wraps' many of the methods found in **metafor** (Viechtbauer 2010) package so that they are consistent with our interface, which we explain in detail in the following subsection. In the (near) future, we plan on expanding our package to include meta-analysis routines for network meta-analysis and other advanced meta-analytic methods.

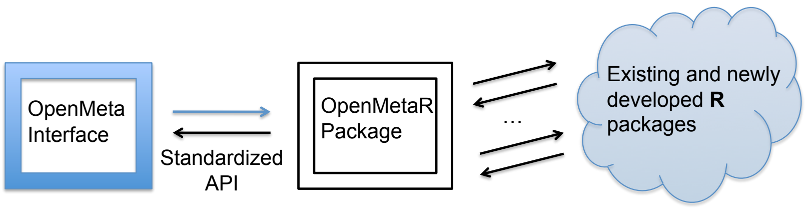


Figure 4: The **openmetar** package is a liason between the UI and the underlying methods, which themselves may live in arbitrary packages. **openmetar** provides a coherent interface (i.e., API) for communication between the 'front' and 'back'-ends; new R methods can be 'plugged-in' to the program by wrapping them in the **openmetar** package, as demonstrated in Section 4.2.

In meta-analysis, there are a few basic data *outcome types* that can be pooled quantitatively. These include: binary outcomes, in which one of two possibilities is realized (e.g., dead/alive); continuous outcomes, which are arbitrary scalars capturing some measured quality (e.g., pain, as expressed on some scale); and diagnostic outcomes, which assess the performance of a diagnostic test (e.g., the sensitivity and specificity of a test). These data types are associated with different types of 'raw data' from which outcomes are calculated. Characterizing the space of data types over which analyses can be performed allows methodologists to write new methods for particular types of data, making assumptions about the parameters these methods will receive.

The interface side of things uses introspection to retrieve a list of available functions in the **openmetar** package for the current data type. It then assumes that method names that begin with, e.g., 'binary' are meta-analysis methods for binary data. We plan on augmenting this for more advanced data types (e.g., network data) in the future. The interface also assumes that there is a function for each of these methods that will return a list of parameter names and types the method requires. These are then used to generate a GUI for the user to input the required values. The GUI elicits these parameters from the user and passes them pack to the method in a dictionary with names corresponding to those given by the `.parameters` method. Figure 3 depicts the described interaction.

4.2. An example

We will now walk through an example of how a methodologist can implement a new meta-analysis method in R, and, so long as they conform to our interface, this method will automatically be available to end users via the (Python) GUI.

In particular, we require that each meta-analysis method that operates on binary data takes as arguments a `BinaryData` S4 object and a named list of parameters (similiarly, continuous meta-analysis methods are assumed to accept a `ContinuousData` object and a parameters list).[2] By distilling the elemental arguments sufficient to run meta-analyses, the GUI can remain agnostic with respect to what method it is invoking, as it needs only to know that it must pass these two arguments. The parameters, of course, will be specific to the meta-

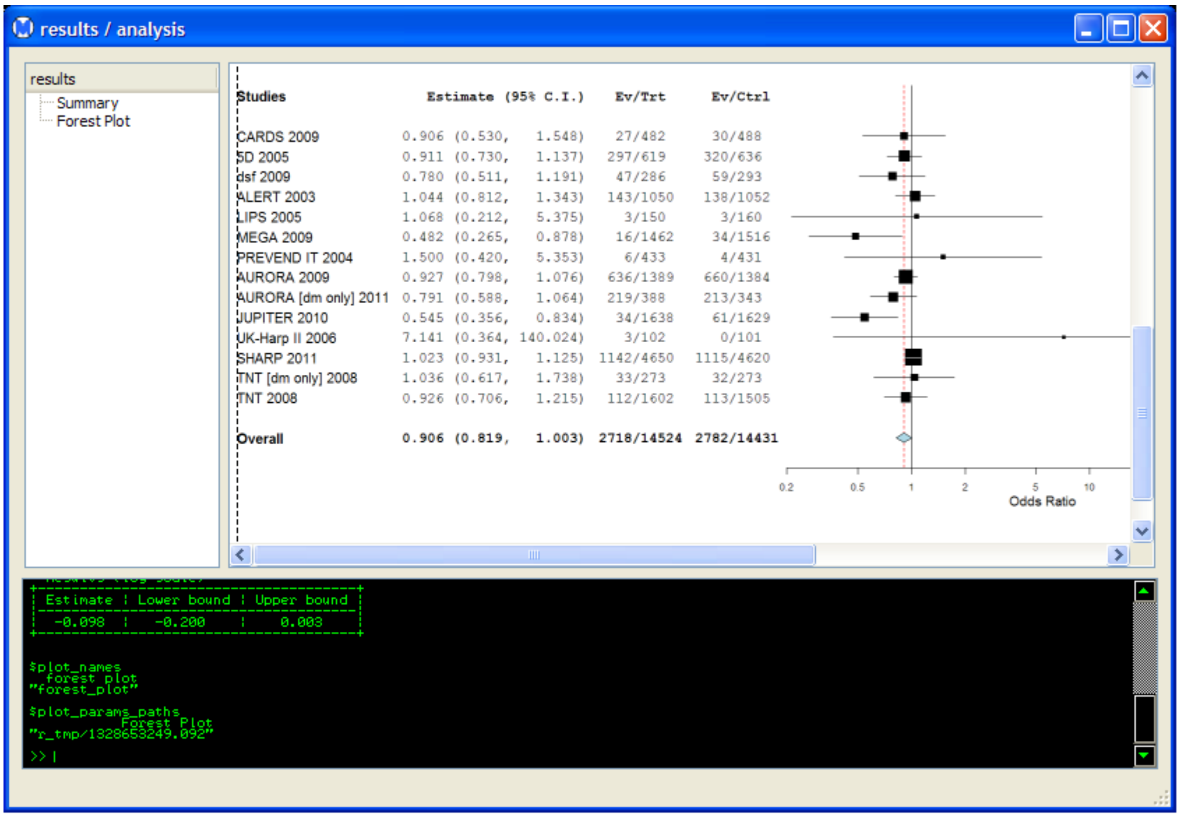
[2]We use S4 as our object system rather than S3 because the former is a more fully realized instantation of an object-orientation system than latter, in our opinion (e.g., S4 allows for inheritance, which we exploit). In any case, we do not think that this choice will much affect methodologists implementing new methods in the system.

```
0 binary.random <- function(binary.data, params) {
1    if (!("BinaryData" %in% class(binary.data))) stop("Binary data expected.")
2    res <- rma.uni(yi = binary.data@y, sei = binary.data@SE,
       slab = binary.data@study.names, method = params$rm.method,
       level = params$conf.level, digits = params$digits)
3    degf <- res$k.yi - 1
4    model.titless <- paste("Binary Random-Effects Model (k = ", res$k, ")",
       sep = "")
5    summary.disp <- create.summary.disp(res, params, degf, model.title)
6    forest.path <- paste(params\$fp_outpath, sep = "")
7    plot.data <- create.plot.data.binary(binary.data, params, res)
8    forest.plot(plot.data, outpath=forest.path)
9    images <- c("Forest Plot" = forest.path)
10   plot.names <- c("forest plot" = "forest_plot")
11   results <- list("images" = images, "Summary" = summary.disp,
       "plot_names" = plot.names)
12   results
13 }
```

Figure 5: Code for our implemenation of the *binary random effects* meta-analysis. The main point is that we are able to wrap existing code so as to conform to our interface: the methods are then automatically made available via the UI. We provide detailed exposition, below. Line 1 asserts that the parametric data is of the right type. Line 2 is the call out to the random effects as implemented by the **metafor** package (Viechtbauer 2010). We parse out the parameters required by this method in order to pass them along to the method we are wrapping. The first three of these parameters are assumed to belong to every `BinaryData` object; the latter three are passed in via the `params` object because the associated `.parameters` method (below) requested them, and so the GUI solicited the user for appropriate values. Lines 3–5 assemble some output for the user, in particular Line 5 invokes a method to construct a pretty-printed summary table. This will be passed back to the GUI, which will display it to the user. Lines 6–8 call out to a routine that constructs an image associated with this type of analysis; in particular this is a 'forest plot', a popular graphic in meta-analysis. The `forest.plot` method generates this figure and writes it out to disk. We then return a named list mapping the name of the plot to the path to this generated image file. This image will subsequently be displayed by the GUI (in Python we will read the image from the disk). In this case there is only one image, however we could generate (and return) an arbitrary number of figures in the same way. Finally, in lines 9–11, we package the results in a dictionary (technically, a named list) to be returned and consuemd by the GUI. In particular, there are two fields that must be returned; the aformentioned dictionary of images (mapping titles to image paths) and a list of texts (mapping titles to pretty-printed text). The GUI will take care of displaying these to the user.

analysis method being implemented – we next discuss how we accomodate this requirement by way of an example.

Let us suppose that we are interested in implementing the *random effects* models we previously discussed. In fact, these methods have already been implemented in the **metafor**



Studies	Estimate (95% C.I.)	Ev/Trt	Ev/Ctrl
CARDS 2009	0.906 (0.530, 1.548)	27/482	30/488
5D 2005	0.911 (0.730, 1.137)	297/619	320/636
dsf 2009	0.780 (0.511, 1.191)	47/286	59/293
ALERT 2003	1.044 (0.812, 1.343)	143/1050	138/1052
LIPS 2005	1.068 (0.212, 5.375)	3/150	3/160
MEGA 2009	0.482 (0.265, 0.878)	16/1462	34/1516
PREVEND IT 2004	1.500 (0.420, 5.353)	6/433	4/431
AURORA 2009	0.927 (0.798, 1.076)	636/1389	660/1384
AURORA [dm only] 2011	0.791 (0.588, 1.064)	219/388	213/343
JUPITER 2010	0.545 (0.356, 0.834)	34/1638	61/1629
UK-Harp II 2006	7.141 (0.364, 140.024)	3/102	0/101
SHARP 2011	1.023 (0.931, 1.125)	1142/4650	1115/4620
TNT [dm only] 2008	1.036 (0.617, 1.738)	33/273	32/273
TNT 2008	0.926 (0.706, 1.215)	112/1602	113/1505
Overall	**0.906 (0.819, 1.003)**	**2718/14524**	**2782/14431**

```
+-----------------------------------------+
| Estimate | Lower bound | Upper bound |
|-----------------------------------------|
|  -0.098  |   -0.200    |    0.003    |
+-----------------------------------------+

$plot_names
  forest plot
"forest_plot"

$plot_params_paths
            Forest Plot
"r_tmp/1328653249.092"

>> |
```

Figure 6: A screenshot of the GUI rendering the results of a meta-analysis.

package (Viechtbauer 2010), and so our implementation of random effects meta-analysis for binary data (Figure 5) simply wraps the pre-existing routine for this contained in the **metafor** package. This example illustrates how using R as an analytic back-end allows us to exploit the wealth of existing statistical packages, thereby freeing us from having to reinvent the proverbial wheel. Moreover, contrary to most existing (GUI-based) meta-analysis software, the **metafor** implementation of random effects meta-analysis includes many alternative estimators of between-study variance, beyond the popular DerSimonian-Laird estimator (see Figure 8).

The code itself is relatively straightforward[3] (the reader is encouraged to read through the commented code). We parse out the elements required by the `rma.uni` function, namely the individual effect estimates (`binary.data@y`) and the corresponding standard errors (`binary.data@SE`). Both of these fields are assumed to exist in all `BinaryData` objects. We pass along some other parameters, as provided in the `params` argument. The rest of the routine is concerned with parsing the output of interest from the `rma.uni` function and packaging it for consumption by the GUI. We also generate a forest plot, which is saved to disk at a user-specified path.

The method must return a named list. This list is assumed to contain an 'images' entry, which is a dictionary that maps image names to their paths on disk. The GUI will render

[3]This code is a simplified version of what is actually used for this routine in the **openmetar** package; here we suppress unimportant details for clarity.

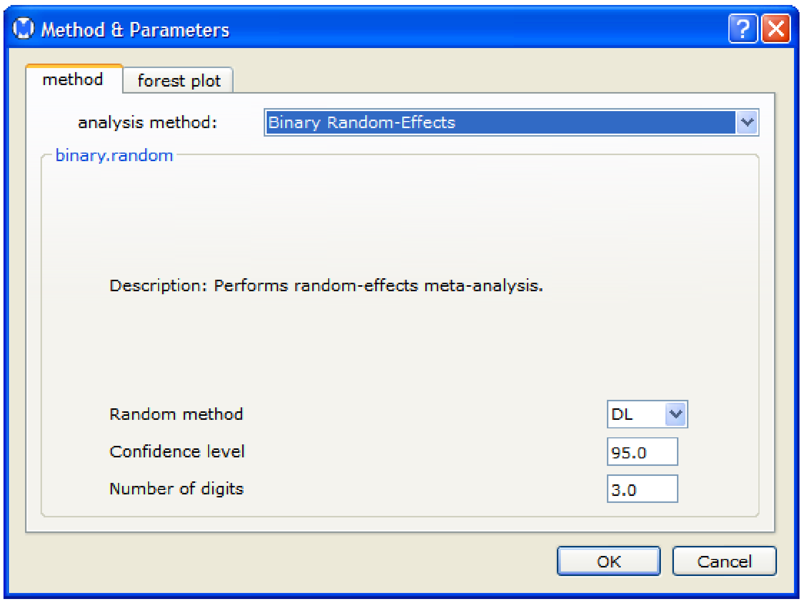
```
0 binary.random.parameters <- function(){
1    rm_method_ls <- c("HE", "DL", "SJ", "ML", "REML", "EB")
2    params <- list("rm.method" = rm_method_ls, "conf.level" = "float",
       "digits" = "float")
3    defaults <- list("rm.method" = "DL", "conf.level" = 95, "digits" = 3)
4    var_order <- c("rm.method", "conf.level", "digits")
5    parameters <- list("parameters" = params, "defaults" = defaults,
       "var_order" = var_order)
6    parameters
7 }
```

Figure 7: The `.parameters` method associated with the `binary.random` function (Figure 5). This method specifies the arguments required by said method, as well as their types. Line 1 specificies the different available estimates for tau^2, discussed above: the user will be able to choose which of these the software usess. Line 2 assembles a complete list of parameters the `binary.random` method requires (and that the user must provide): the estimation method, the confidence level and the number of digits to be displayed. Crucially, this requires that the code here specify what *type* of values each of these parameters may take; this allows the interface code to draw suitable input widgets. In Line 3 the defaults for these are provided. The GUI will automatically set the corresponding parameters to these values, initially; the user can then change them. In 4 we specify the order in which these variables are to appear in the GUI, vertically (see Figure 8). Finally, in line 5 we package these up for consumption by the Python interface code, which will be responsible for rendering an interface that allows the user to impart values for these parameters to the system.

these for the end user. Other entries in the returned list are assumed to correspond to textual output – in this case we only have a 'Summary' field, which contains information about the analysis and the overall (pooled) estimate, etc. An arbitrary number of text fields may be returned. The text will be rendered in the GUI exactly as it would appear at the console, so the programmer is responsible for 'pretty-printing' the output.

There remains the issue that different meta-analysis methods will require different parameters. For example, a random-effects meta-analysis requires the specification of the random effects method to be used, whereas no such parameter is required for fixed-effects analysis. We therefore need a way for R programmers who are implementing meta-analysis methods to communicate to the GUI what parameters are required for their routine. The GUI will then be responsible for rendering an interface that solicits these parameters from users and, ultimately, passes them to the analytic routine.

We accomplish this again by making assumptions about naming conventions; every meta-analysis method is assumed to have a corresponding `.parameters` routine, whose only purpose is to return a named list specifying the argument names and types required by the the associated function. For example, the `binary.random.parameters` method shows the `.parameters` routine for the random effects meta-analysis routine for binary data implemented in Figure 5. Note that the type of each argument must also be specified. For example, the `conf.level` parameter is a float, whereas the `rm.method` argument is a categorical string. In the latter case, the values the argument can assume are enumerated in a list, which will be presented



analysis method: Binary Random-Effects

binary.random

Description: Performs random-effects meta-analysis.

Random method DL

Confidence level 95.0

Number of digits 3.0

Figure 8: The GUI that is automatically built and presented to the end-user using the values specified by the `binary.random.parameters` (Figure 7).

to the user as a standard combo box (see Figure 8).

We allow the author of the method to optionally specify default values for their parameters using the `defaults` list, as shown in Figure 7; here, for example, we set the default confidence interval to 95. We also allow the author to optionally specify the order in which the parameters are presented to the user (e.g., in Figure 7, the `rm.method` argument will be at the top of the GUI, as shown in Figure 8).

An example of the GUI rendering the output returned in the last line of the routine shown in Figure 5 is displayed in Figure 6. Note also that in the bottom panel of the results window there is a small console; this provides direct access to the R environment. More advanced users may examine their results via the console, if they would prefer, and non-technical users can simply ignore it.

An additional part of the architecture not demonstrated by this example is the ability for the method author to express necessary conditions for their function. For example, a meta-analysis method may require at least three studies (i.e., individual point estimates) to run. The author of such a method can enforce this requirement by implementing a `.is.feasible` function associated with their method. This function is passed the current data object, which can then be inspected to see if the corresponding meta-analysis routine is feasible given the provided data. The function is returns `True` if the function can be run and `False` otherwise. Furthermore, we allow the implementer of methods to optionally provide more informative names for both the method and parameters, as well as descriptions for both. If these are available, the GUI will retrieve and display them. To recapitulate; the GUI requests which methods for the current data type are available by using introspection and making assumptions about naming conventions, and generates an interface for soliciting user parameters for these methods based on the corresponding `.parameters` function. Methods are assumed to return pointers to image files and pretty-printed text as output.

5. Discussion

In meta-analysis, the methodologists who invent and implement cutting-edge meta-analysis methods often do so in R. Meta-analysts are often non-technical and therefore tend to use GUI-driven programs instead of R to perform their analyses. Thus there is a gap between the statistical methods that ought to be used and the methods that are used due to the limited availability of the former in GUI-based programs. It is difficult for software developers to continuously implement and integrate the latest statistical methods into specialized programs, particularly when the methods are fast-evolving. This problem is compounded by the fact that such specialized GUI-based programs are written in general programming languages, not in specialized statistical languages like R.

To address this issue for the case of meta-analysis software, we have presented a framework for using R as the underlying computational/statistical engine, while the GUI is written in a different language more suited to GUI programming (in our case, Python). We have outlined our framework for extending the methods available in the GUI, which does not require directly writing GUI code. Thus new methods developed by methodologists can be made available to a more general, non-technical set of meta-analysts with minimal effort – the methods need only be wrapped so as to conform to our architecture. In our case, we have opted to use this strategy in developing a desktop application with the QT toolkit. However because of our strict separation of the analytic and UI componenents, one could re-implement the UI generation (following the described API) in, e.g., a web-interface with relative ease.

While meta-analysis was the driving example in this case, we believe this framework of using R as the underlying computational engine and a different language for the GUI could be of use in many mathematically driven application areas, particularly when the target users of new methods may be non-technical. Such an approach is suitable whenever the problem space (e.g., meta-analysis methods) is more conveniently expressed in a statistical language such as R but there is a need for an intuitive GUI for end-users. Some examples of such areas include: decision theory, machine learning, and network analysis. Given that this strategy is likely widely applicable, it would perhaps be useful in future work to extract the UI generating functionality into a stand-alone Python library, which could attempt to build UIs (automatically) for any R package that conformed to the API. Admittedly, however, such a general package would require careful design.

6. Code

OpenMeta-Analyst is a completely open source project. The source code (both Python and R) is available on GitHub at: `http://github.com/bwallace/OpenMeta-analyst-`.

Compiled 'binary' releases of the program are available to Windows and OS X users at `http://tuftscaes.org/open_meta/`.

Acknowledgments

This work was made possible by grant R01 HS018574-01 from the Agency for Healthcare Research and Quality (AHRQ).

References

Airy GB (1875). *On the Algebraical and Numerical Theory of Errors of Observations and the Combination of Observations.* Macmillan.

Arthur W, Bennett W, Huffcutt AI, Arthur JW, Bennett JW (2001). *Conducting Meta-Analysis Using SAS.* Lawrence Erlbaum.

Bax L, Yu LM, Ikeda N, Tsuruta H, Moons KGM (2006). "Development and Validation of **MIX**: Comprehensive Free Software for Meta-Analysis of Causal Research Data." *BMC Medical Research Methodology*, **6**(1), 50.

Borenstein M, Rothstein H, Cohen J (2005). ***Comprehensive Meta-Analysis**, Version 2: A Computer Program for Research Synthesis.* Englewood.

Clarke M, Oxman A (2000). "Review Manager (**RevMan**)."

DerSimonian R, Laird N (1986). "Meta-Analysis in Clinical Trials." *Controlled Clinical Trials*, **7**(3), 177–188.

Fisher RA (1925). *Statistical Methods for Research Workers.* Genesis Publishing.

Fox J (2005). "The R Commander: A Basic-Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL http://www.jstatsoft.org/v14/i09/.

Glass GV (1976). "Primary, Secondary, and Meta-Analysis of Research." *Educational Researcher*, **5**(10), 3–8.

Gotzsche PC, Hrobjartsson A, Maric K, Tendal B (2007). "Data Extraction Errors in Meta-Analyses that Use Standardized Mean Differences." *Journal of the American Medical Association*, **298**(4), 430.

Kontopantelis E, Reeves D (2009). "**MetaEasy**: A Meta-Analysis Add-In for Microsoft **Excel**." *Journal of Statistical Software*, **30**(7), 1–25. URL http://www.jstatsoft.org/v30/i07/.

Kulik JA, Kulik CLC (1989). "Meta-Analysis in Education." *International Journal of Educational Research*, **13**(3), 221–340.

Lau J (1997). "**Meta-Test** Version 0.6: A Program to Conduct Statistical Analysis of Diagnostic Test Data."

Lipsey MW, Wilson DB (2001). *Practical Meta-Analysis.* Sage Publications.

Lumley T (2009). ***rmeta**: Meta-Analysis.* R package version 2.16, URL http://CRAN.R-project.org/package=rmeta.

Moriera W, Warnes GR (2004). "**rpy**: A Robust Python Interface to the R Programming Language." URL http://rpy.sourceforge.net/.

Osenberg CW, Sarnelle O, Goldberg DE (1999). "Meta-Analysis in Ecology: Concepts, Statistics, and Applications." *Ecology*, **80**(4), 1103–1104.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

River Bank Computing (2012). "**PyQT**." URL `http://www.riverbankcomputing.com/software/pyqt`.

Rosenberg MS, Adams DC, Gurevitch J (1997). ***Metawin**: Statistical Software for Meta-Analysis with Resampling Tests*. Sinauer Associates.

Schwarzer G (2012). ***meta**: Meta-Analysis with R*. R package version 2.1-0, URL `http://CRAN.R-project.org/package=meta`.

Simpson RJS, Pearson K (1904). "Report on Certain Enteric Fever Inoculation Statistics." *The British Medical Journal*, **2**(2288), 1243–1246.

Sterne JAC (2009). *Meta-analysis in Stata: An Updated Collection from the Stata Journal*. StataCorp LP.

Stroup DF, Berlin JA, Morton SC, Olkin I, Williamson GD, Rennie D, Moher D, Becker BJ, Sipe TA, Thacker SB, others (2000). "Meta-Analysis of Observational Studies in Epidemiology: A Proposal for Reporting." *Journal of the American Medical Association*, **283**(15), 2008.

Turner RM, Omar RZ, Yang M, Goldstein H, Thompson SG (2000). "A Multilevel Model Framework for Meta-analysis of Clinical Trials with Binary Outcomes." *Statistics in Medicine*, **19**(24), 3417–3432.

Viechtbauer W (2005). "Bias and Efficiency of Meta-Analytic Variance Estimators in the Random-Effects Model." *Journal of Educational and Behavioral Statistics*, **30**(3), 261.

Viechtbauer W (2010). "Conducting Meta-Analyses in R with the **metafor** Package." *Journal of Statistical Software*, **36**(3), 1–48. URL `http://www.jstatsoft.org/v36/i03/`.

Wallace BC, Schmid CH, Lau J, Trikalinos TA (2009). "**Meta-Analyst**: Software for Meta-analysis of Binary, Continuous and Diagnostic Data." *BMC Medical Research Methodology*, **9**(1), 80.

Zamora J, Abraira V, Muriel A, Khan K, Coomarasamy A (2006). "**Meta-DiSc**: A Software for Meta-Analysis of Test Accuracy Data." *BMC Medical Research Methodology*, **6**(1), 31.

Affiliation:

Byron C. Wallace
Department of Computer Science
Tufts University

Center for Clinical Evidence Synthesis
Tufts Medical Center
Boston, MA, United States of America
E-mail: `byron.wallace@tufts.edu`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the American Statistical Association `http://www.amstat.org/`

Volume 49, Issue 5 *Submitted:* 2010-11-01
June 2012 *Accepted:* 2011-12-20