Automating Reproducible, Collaborative Clinical Trial Document Generation with the listdown Package

by Michael Kane, Xun Jiang, and Simon Urbanek

Abstract The conveyance of clinical trial explorations and analysis results from a statistician to a clinical investigator is a critical component of the drug development and clinical research cycle. Automating the process of generating documents for data descriptions, summaries, exploration, and analysis allows the statistician to provide a more comprehensive view of the information captured by a clinical trial, and efficient generation of these documents allows the statistican to focus more on the conceptual development of a trial or trial analysis and less on the implementation of the summaries and results on which decisions are made. This paper explores the use of the **listdown** package for automating reproducible documents in clinical trials that facilitate the collaboration between statisticians and clinicians as well as defining an analysis pipeline for document generation.

Background and Introduction

The conveyance of clinical trial explorations and analysis results from a statistician to a clinical investigator is an often overlooked but critical component to the drug development and clinical research cycle. Graphs, tables, and other analysis artifacts are at the nexus of these collaborations. They facilitate identifying problems and bugs in the data preparation and processing stage, they help to build an intuitive understanding of mechanisms of disease and their treatment, they elucidate prognostic and predictive relationships, they provide insight that results in new hypotheses, and they convince researchers of analyses testing hypotheses.

Despite their importance, the process of generating these artifacts is usually done in an ad-hoc manner. This is partially because of the nuance and diversity of the hypotheses and scientific questions being interrogated and, to a lesser degree, the variation in clinical data formatting. The usual process usually has a statistician providing a standard set of artifacts, receiving feedback, and providing updates based on feedback. Work performed for one trial is rarely leveraged on others, and as a result, a large amount of work needs to be reproduced for each trial.

There are two glaring problems with this approach. First, each analysis of a trial requires a substantial amount of error-prone work. While the variation between trials means some work needs to be done for preparation, exploration, and analysis, many aspects of these trials could be better automated resulting in greater efficiency and accuracy. Second, because this work is challenging, it often occupies the majority of the statisticians' effort. Less time is spent on trial design and analysis, and then this portion is taken up by a clinician who often has less expertise with the statistical aspects of the trial. As a result, the extra effort spent on processing data undermines statisticians' role as a collaborator and relegates them to service provider. Need tools leveraging existing work to more efficiently provide holistic views on trials will result in less effort and more accurate and comprehensive trial design and analysis.

The richness of the R Core Team (2012) package ecosystem, particularly with its emphasis on analysis, visualization, reproducibility, and dissemination makes the goal of creating these tools for clinical trials feasible. Generation of tables is supported by packages including **tableone** (Yoshida and Bartel, 2020), **gt** (Iannone et al., 2020), **gtsummary** (Sjoberg et al., 2020). Visualization is achieved using package including **ggplot2** (Wickham, 2016) and **survminer** (Kassambara et al., 2020). We can even provide interactive presentations of data with **DT** (Xie et al., 2020), **plotly** (Sievert, 2020), and **trelliscopejs** (Hafen and Schloerke, 2020).

It should also be realized that work building on these tools for clinical trial data is already in process. The **greport** (Harrell Jr, 2020) package provides graphical summaries for clinical trials and has been used in conjunction with **rmarkdown** (Allaire et al., 2020) to produce specific trial report types with a specified format.

Using listdown for programmatic, collaborative clinical trial document generation

The **listdown** package (Kane et al., 2020) was recently released to automate the process of generating reproducible (RMarkdown) documents. Objects derived from a summary, exploration, or analysis are stored hierarchically in an R `list`, which defines the structure of the document. These objects are referred to as *computational components* since they are derived from computation, as opposed to prose, which makes up the *narrative components* of a document.

The computational components capture and structure the objects to be presented. Describing how the objects will be presented and how the document will be rendered is handled through the creation of a `listdown` object. The separation between how computational components are created and how they are shown to a user provides two advantages. First, it decouples the data processing and analysis from its exploration and visualization. For compute-intensive analyses, this separation is critical for avoiding redundant computations for small changes in the presentation. It also discourages putting compute-intensive code into RMarkdown documents. Second, it provides the flexibility to quickly change how a computational component is visualized or summarized or even how a document is rendered. This makes transitioning from an interactive .html document to a static .pdf document significantly easier than substituting functions and parameters in an R Markdown document.

The package has been found to be particularly useful in the reporting and research of clinical trial data. In particular, the package has been used for server collaborations focusing on either the analysis of past trial data to formulate a new trial or in trial monitoring where trial telemetry (enrollment, responses, etc.) is reported, and initial analyses are conveyed to a clinician. The associated presentations require very little context since clinicians often have as good an understanding of the data collected as that of the statistician's meaning narrative components are not needed. At the same time, a large number of hierarchical, heterogeneous artifacts (tables and multiple types of plots) can be automated where manual creation of RMarkdown documents would be inconvenient and inefficient.

The rest of this document describes concepts implemented in the **listdown** package for automated, reproducible document generation and shows its use with a simplified, synthetic clinical trial data set whose variables are typical of a non-small cell lung cancer trial. The data set comes from the **forceps** (Kane, 2020) package. As of the time this document was written, the package is under development and is not available on CRAN. However, it can be installed as follows.

```
devtools::install_github("kaneplusplus/forceps")
```

The following section uses the trial data to construct a pipeline for document generation. We note that both the data and the pipeline are simple when compared to most analyses of this type. However, it is sufficient to illustrate accompanying concepts, and both the analyses and concepts translate readily to real-world applications. A final section discusses the use of the package and its current direction.

Constructing a pipeline for document generation

The process of analyzing data can be described using the classic waterfall model of Benington (1983) where the output (the analysis presentation or service) is dependent on a sequence of tasks that come before it. This dependency structure means that if a problem is detected in a given stage of the production of the analysis, all downstream parts must be rerun to reflect the change. A graphical depiction of the waterfall model, specific to data analyses (clinical or otherwise) is shown in Figure 1. Note that data exploration and visualization are an integral part of all stages of the production and are often the means for identifying issues and refining analyses.

As explained in the previous section, we are going to implement a simple analysis pipeline. The data acquisition and preprocessing steps are handled by importing data sets from the **forceps** package and using some of the functions implemented in the package to create a single trial data set, thereby de-emphasizing these components in the pipeline. While these steps are critical, the emphasis of this paper is the incorporation of the **listdown** package into the later stages.

Data acquisition and preprocessing

Data acquisition refers to the portion of the analysis pipeline where the data is retrieved from some managed data store for integration into the pipeline. These data sets may be retrieved as tables from a database, case reports, Analysis Data Model (ADaM) data formatted according to the Clinical Data Interchange Standards Consortium (CDISC) (CDI, 2020), Electronic Health Records, or other clinical Real World Data (RWD) formats. These data are then transformed to a format appropriate for analysis.

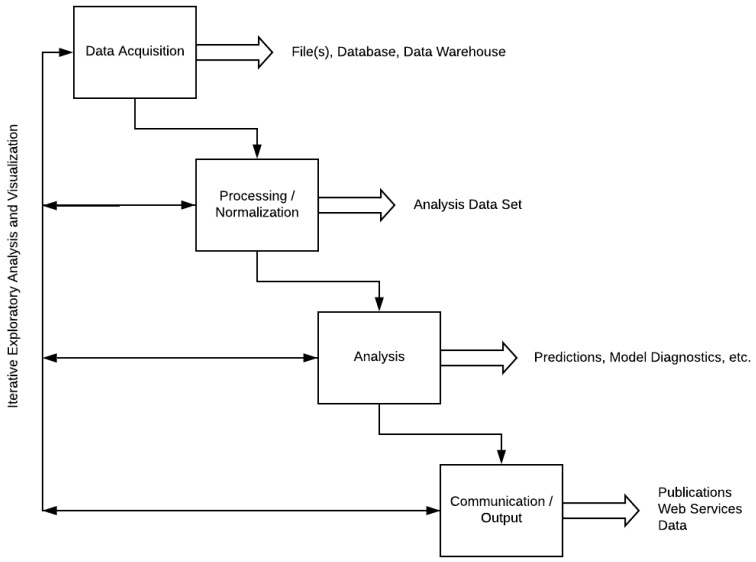

Figure 1: The data analysis waterfall.

In our simple example, this is accomplished by loading data corresponding to trial outcomes, patient adverse events, patient biomarkers, and patient demography and transforming them into a single data set with one row per patient and one variable per column using the **forceps** and **dplyr** (Wickham et al., 2020) packages. The data also includes longitudinal adverse event information, which will is stored as a nested data frame in the `ae_long` column of the resulting data set.

```
library(forceps)
library(dplyr)

data(lc_adsl, lc_adverse_events, lc_biomarkers, lc_demography)

lc_trial <- consolidate(
  list(adsl = lc_adsl,
       adverse_events = lc_adverse_events %>%
         cohort(on = "usubjid", name = "ae_long"),
       biomarkers = lc_biomarkers,
       demography = lc_demography %>%
         select(-chemo_stop)
  ),
  on = "usubjid")

lc_trial
```

```
#> # A tibble: 558 x 18
#>    usubjid best_response  pfs_days pfs_censor os_days os_censor chemo_stop arm
#>      <dbl> <chr>             <dbl>      <dbl>   <dbl>     <dbl> <chr>      <chr>
#>  1    1003 Stable Disease      101          1     233         1 adverse e~ stan~
#>  2    1005 Complete Resp~       78          0     184         1 adverse e~ stan~
#>  3    1006 Progressive D~      253          1     333         1 treatment~ stan~
#>  4    1009 Partial Respo~      130          0     643         0 <NA>       trea~
#>  5    1014 Complete Resp~       41          1     116         1 adverse e~ trea~
#>  6    1018 Partial Respo~      194          1     423         1 treatment~ stan~
#>  7    1023 Stable Disease       49          1     337         1 adverse e~ stan~
#>  8    1025 Stable Disease       95          1     589         1 adverse e~ stan~
#>  9    1030 Complete Resp~      351          1     688         1 adverse e~ stan~
#> 10    1033 Complete Resp~       33          1     125         1 treatment~ stan~
#> # ... with 548 more rows, and 10 more variables: ae_count <int>,
#> #   ae_long <list>, egfr_mutation <chr>, smoking <chr>, ecog <chr>,
#> #   prior_resp <chr>, site_id <int>, sex <chr>, refractory <lgl>, age <dbl>
```

Analysis defining and structuring the computational components

The next step is to define the computational components as a hierarchical, named list of objects that will be presented. These objects are generally derived from the exploratory, descriptive, and analysis stages and are presented as visualizations and tables. This example will focus on the exploratory and descriptive portions. We will start by defining a new function `class_and_tag()`, which takes an object and prepends a class designation to the objects along with optionally associating the object with a set of attributes. This extra information will be used later in order to dispatch to the proper functions responsible for presenting the objects in a resulting R Markdown document. If no such information is given, then the object will be presented as it usually would in the document. It should also be noted that `class_and_tag()` is provided for convenience here and will be available in subsequent package versions.

The next step is to define the computational components that will be presented. Our simple example will contain three sections: Outcome Information, Adverse Events, Biomarkers. The Adverse Events and Biomarkers sections will each show summary tables of one of the variables from those data. The Outcome Information section will be composed of two subsections, with the first (Best Response) providing a summary of the best response by the arm and the second (Overall Survival) showing a survival plot by arm of the overall survival. The call to `ld_cc_dendro()` at the end of the example provides of dendrogram of the hierarchical structure.

```
library(listdown)

trial_summary <- list(
  `Outcome Information` = list(
    `Best Response` = lc_trial %>%
      select(best_response, arm) %>%
      class_and_tag("summary_table", by = "arm"),
    `Overall Survival` = lc_trial %>%
      select(os_days, os_censor, arm) %>%
      class_and_tag("survival_plot",
                    time = "os_days",
                    censor = "os_censor",
                    x = "arm")),
  `Adverse Events` = lc_trial %>%
    select(best_response) %>%
    class_and_tag("summary_table"),
  `Biomarkers` = list(
    `EGFR` = lc_trial %>%
      select(egfr_mutation) %>%
      class_and_tag("summary_table")
  )
)

ld_cc_dendro(trial_summary)

#>
#> trial_summary
#>   |-- Outcome Information
#>    |-- Best Response
#>    |  o-- object of type(s):summary_table tbl_df tbl data.frame
#>    o-- Overall Survival
#>       o-- object of type(s):survival_plot tbl_df tbl data.frame
#>   |-- Adverse Events
#>   |  o-- object of type(s):summary_table tbl_df tbl data.frame
#>   o-- Biomarkers
#>    o-- EGFR
#>       o-- object of type(s):summary_table tbl_df tbl data.frame
```

Communicating results

The objects have been constructed and they have been structured. The next step is to define how they will be presented. This is done by creating two functions `make_summary` and `make_survival_plot`. The former takes a `data.frame` and uses the **gtsummary** package to summarize the results. If the `data.frame` includes an attribute named `by` and denoting a valid variable in the data set, then the

summary is created conditionally on that variable. The latter function also takes a `data.frame` and uses attributes to denote variables on which a Kaplan-Meier plot can be constructed using the **survival** (Terry M. Therneau and Patricia M. Grambsch, 2000) and **survminer** packages. The functions are written to a file called `decorators.r` and will be used by the R Markdown document to render the final document.

```
library(gtsummary)
library(survival)
library(survminer)
library(dplyr)

make_summary <- function(x) {
  by <- attributes(x)$by
  tbl_summary(x, by = by)
}

make_survival_plot <- function(.x) {
  att <- attributes(.x)
  x <- ifelse(is.null(att$x), "1", att$x)
  form <- sprintf("Surv(%s, %s) ~ %s", att$time, att$censor, x) %>%
    as.formula()
  fit <- surv_fit(form, data = as.data.frame(.x))
  ggsurvplot(fit, data = as.data.frame(.x))
}
```

The next step is to connect the computational components to the functions that will present them using the `listdown()` function. First, the computational components are stored in the `trial_summary` object are written to the disk. The resulting R Markdown document will read it in based on the `load_cc_expr` argument. Along with reading in the data, initialization in the R Markdown document will include sourcing the `decorators.r` file previously written to disk. This is handled with the `init_expr` argument. The `decorators` argument takes a list where the name specifies the class and the list element corresponds to a function that will present objects of the specified class. For example, `summary_table` objects are sent to the `make_summary()` function for presentation. This allows us to connect objects to their appropriate function to process and present them. Finally, `echo` and `message` chunk options are set to `FALSE` so that the code and associated messages will not appear in the final rendered document. The last line of code below displays the first 12 lines of R Markdown code chunks as they will appear in the corresponding document.

```
saveRDS(trial_summary, "cc.rds")

ld <- listdown(load_cc_expr = readRDS("cc.rds"),
               init_expr = source("decorators.r"),
               decorator = list(summary_table = make_summary,
                                survival_plot = make_survival_plot),
               echo = FALSE,
               message = FALSE)

ld_make_chunks(ld)[1:12]

#>  [1] ""
#>  [2] "```{r echo = FALSE, message = FALSE}"
#>  [3] "source(\"decorators.r\")"
#>  [4] ""
#>  [5] "cc_list <- readRDS(\"cc.rds\")"
#>  [6] "```"
#>  [7] ""
#>  [8] "# Outcome Information"
#>  [9] ""
#> [10] "## Best Response"
#> [11] ""
#> [12] "```{r echo = FALSE, message = FALSE}"
```

The last step creates the R Markdown header, writes it along with the R code chunks to a file named `"simple-data-trial-summary.rmd"` and knits the file with the **knitr** package (Xie, 2020) to a pdf document, per the `output` argument of the `ld_rmarkdown_header()` function. `md_header` is a `yml`

object making it easy to control how the document is rendered. The code to generate the document along with the resulting R Markdown and output file constitute a reproducible workflow that can be quickly iterated upon and adapted for different types of explorations, summaries, and analyses.

```
library(knitr)

md_header <- ld_rmarkdown_header("Fake Data Trial Summary",
                                 output = "pdf_document")

ld_write_file(
  rmd_header = as.character(md_header),
  ld = ld_make_chunks(ld),
  file_name = "simple-data-trial-summary.rmd")

knit("simple-data-trial-summary.rmd")
```

Direction

The **listdown** package has been successfully used for collaborations on several clinical trial analyses. The package works particularly well when creating large, navigable sets of summaries and visualizations. Current work focuses on two areas. First is the construction of standard decorators. By packaging decorators and associated functionality, presentations can be made rich as they are developed over time. This standardization also makes it easier to leverage work in configuring chunks so that they can be made more aesthetic by default. In addition, we have been working on abstract and formalize the notion of document composition. In the example presented, the aggregation of the header and R code chunks into a file was sufficient for generating a document. However, the composition of other outputs is also supported. For example, the top-level names of the trial summary could just as easily designate tabs on a web page or other target. The notion of a composer would allow a user to target a greater variety of output types, better suiting the application under consideration and the target audience for the presentation.

Bibliography

Clinical data interchange standards consortium. https://www.cdisc.org/, 2020. Accessed: 2020-10-25. [p2]

J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2020. URL https://github.com/rstudio/rmarkdown. R package version 2.5. [p1]

H. D. Benington. Production of large computer programs. *Annals of the History of Computing*, 5(4): 350–361, 1983. doi: 10.1109/MAHC.1983.10102. [p2]

R. Hafen and B. Schloerke. *trelliscopejs: Create Interactive Trelliscope Displays*, 2020. URL https://CRAN.R-project.org/package=trelliscopejs. R package version 0.2.5. [p1]

F. E. Harrell Jr. *greport: Graphical Reporting for Clinical Trials*, 2020. URL https://CRAN.R-project.org/package=greport. R package version 0.7-2. [p1]

R. Iannone, J. Cheng, and B. Schloerke. *gt: Easily Create Presentation-Ready Display Tables*, 2020. URL https://CRAN.R-project.org/package=gt. R package version 0.2.2. [p1]

M. J. Kane. *forceps: A grammar for manipulating clinical trial data*, 2020. URL https://github.com/kaneplusplus/forceps. R package version 0.0.2. [p2]

M. J. Kane, X. T. Jiang, and S. Urbanek. On the programmatic generation of reproducible documents, 2020. [p2]

A. Kassambara, M. Kosinski, and P. Biecek. *survminer: Drawing Survival Curves using 'ggplot2'*, 2020. URL https://CRAN.R-project.org/package=survminer. R package version 0.4.8. [p1]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p1]

C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL https://plotly-r.com. [p1]

D. D. Sjoberg, M. Curry, M. Hannum, K. Whiting, and E. C. Zabor. *gtsummary: Presentation-Ready Data Summary and Analytic Result Tables*, 2020. URL https://CRAN.R-project.org/package=gtsummary. R package version 1.3.5. [p1]

Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3. [p5]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p1]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL https://CRAN.R-project.org/package=dplyr. R package version 1.0.2. [p3]

Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2020. URL https://yihui.org/knitr/. R package version 1.30. [p5]

Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2020. URL https://CRAN.R-project.org/package=DT. R package version 0.16. [p1]

K. Yoshida and A. Bartel. *tableone: Create 'Table 1' to Describe Baseline Characteristics with or without Propensity Score Weights*, 2020. URL https://CRAN.R-project.org/package=tableone. R package version 0.12.0. [p1]

Michael Kane
Yale University
60 College Street
New Haven, CT 06510, USA
michael.kane@yale.edu

Xun Jiang
Amgen Inc.
One Amgen Center Drive
Thousand Oaks, CA 91320-1799, USA
xunj@amgen.com

Simon Urbanek
The University of Aukland
38 Princes Street
Auckland Central, Aukland 1010, NZ
s.urbanek@auckland.ac.nz