multiverse: Multiplexing Alternative Data Analyses in R Notebooks

Abhraneel Sarma
abhraneel@u.northwestern.edu
Northwestern University
Evanston, Illinois, USA

Alex Kale
kalea@uchicago.edu
University of Chicago
Chicago, IL, USA

Michael Moon
michael.moon@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Nathan Taback
nathan.taback@utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Fanny Chevalier
fanny@dgp.toronto.edu
University of Toronto
Toronto, Ontario, Canada

Jessica Hullman
jhullman@northwestern.edu
Northwestern University
Evanston, Illinois, USA

Matthew Kay
mjskay@u.northwestern.edu
Northwestern University
Evanston, Illinois, USA

ABSTRACT

There are myriad ways to analyse a dataset. But which one to trust? In the face of such uncertainty, analysts may adopt multiverse analysis: running all reasonable analyses on the dataset. Yet this is cognitively and technically difficult with existing tools—how does one specify and execute all combinations of reasonable analyses of a dataset?—and often requires discarding existing workflows. We present `multiverse`, a tool for implementing multiverse analyses in R with expressive syntax supporting existing computational notebook workflows. `multiverse` supports building up a multiverse through local changes to a single analysis and optimises execution by pruning redundant computations. We evaluate how `multiverse` supports programming multiverse analyses using (a) principles of cognitive ergonomics to compare with two existing multiverse tools; and (b) case studies based on semi-structured interviews with researchers who have successfully implemented an end-to-end analysis using `multiverse`. We identify design tradeoffs (e.g. increased flexibility versus learnability), and suggest future directions for multiverse tool design.

CCS CONCEPTS

• **Software and its engineering** → **Designing software**; • **Human-centered computing** → User studies; *Interaction paradigms*.

KEYWORDS

Multiverse Analysis, robust statistical analysis, cognitive dimensions of notations

CHI '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Abhraneel Sarma, Alex Kale, Michael Moon, Nathan Taback, Fanny Chevalier, Jessica Hullman, and Matthew Kay. 2023. multiverse: Multiplexing Alternative Data Analyses in R Notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Statisticians and meta-science researchers propose *multiverse analysis* [56] and related statistical procedures [47, 55] as a way of making exploratory data analysis (EDA) more robust and transparent. Multiverse analysis entails implementing many combinations of alternative data analysis decisions that a researcher deems reasonable and reporting on the results of these multiple analyses in aggregate. These methods are motivated by the idea that undisclosed *researcher degrees of freedom*—flexibility in analytic decision-making that researchers exercise during EDA—contribute to the replication crisis and increase the probability of erroneous findings [5, 54]. By specifying alternative analyses in a principled way, multiverse analysis can expose the impact of researchers' choices by presenting estimates for each of many possible combinations of analysis decisions [16].

In designing tools to support authoring multiverse analyses, it is important to consider the existing workflows of analysts and data science workers and how such tools might fit into these workflows. Studies investigating the practices of data science workers [31, 32, 49] have found that experts engage in extensive iteration and exploration in order to identify appropriate ways of implementing and executing a data analysis. Literate programming [35] environments such as computational notebooks are well suited for such workflows, as analysts can easily attempt different approaches through trial and error and get immediate feedback.

In this paper we describe the design and implementation of `multiverse`, *an R library designed for analysts to implement multiverse analyses in RMarkdown*, and evaluate how `multiverse` supports users in programming multiverse analyses. The `multiverse`

R library extends syntax in R to allow users to declare alternative analysis options through local changes in code as *branches*, and integrates with the RStudio[1] IDE to support the immediate feedback that users expect from a computational notebook. The design of `multiverse` is grounded in prior empirical work on data analysts' workflows and addresses gaps we identify in state-of-the-art solutions, such as a lack of support for analysts to quickly alternate between writing and evaluating provisional code as they specify a multiverse (§2). Based on this, we articulate five design requirements for multiverse programming interfaces that inform our solution (§3). Our approach is further informed by feedback on an early prototype of the package gathered during an evaluation with analysts.

We evaluate how `multiverse` enables users to program a multiverse analysis in two ways. First, to better understand the implications of different multiverse tool designs on usability, we compare it against three multiverse programming interfaces—`multiverse`, Boba [37], and mverse (a tool built on top of `multiverse`)—using the cognitive dimensions of notations [6, 24] and the gulfs of execution and evaluation [28, 44]. This evaluation (§4) surfaces potential trade-offs in the design space of multiverse APIs, such as providing a flexible syntax versus a more constrained but familiar syntax. Second, we conduct semi-structured interviews with three researchers, who each discovered the library on their own and had successfully implemented an end-to-end analysis with `multiverse`, which we present as case studies (§5). These studies help us understand how researchers learn the syntax of the API and use it to specify their analysis. We find that users are able to use the syntax to not only specify their analysis, but also to combine aspects of the syntax to build new usage patterns, suggesting that the `multiverse` API is flexible and expressive enough to construct complex multiverse-style analyses. We discuss the limitations of current multiverse APIs, such as the lack of effective debugging support, and consider how a stepwise debugger or realtime visualisations can be adapted to assist with debugging multiverse analyses (§6). In addition, we identify a need for theoretical guidance on how to declare principled multiverse analyses, and reflect on the potential of multiverse tools to support ideas of transparent statistical reporting [20].

2 RELATED WORK

2.1 Multiverse Analysis

Exploratory data analysis (EDA), by definition, involves data analysts exploring various strategies for data processing, manipulation and/or modeling that can lead to different results, and maybe different conclusions. However, what is deemed to be a reasonable strategy may vary substantially across different scientists and researchers. Illustrating this phenomena, when a crowdsourced study presented 29 different teams of researchers with the same data set and research question, 29 different analysis approaches were observed [53]. Even though many scientific studies present a single analysis on a dataset, researchers often have flexibility in how to conduct a single analysis on their data (see Figure 1a), a phenomenon referred to as *researcher degrees of freedom* [23, 54]. However, for most scientific studies, only a single analysis is reported, and

[1]https://rstudio.com/

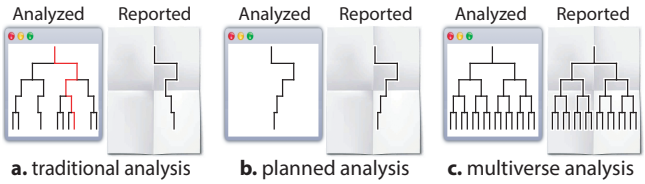

Figure 1: Reporting strategies, from least to most transparent: a) traditional analysis with undisclosed flexibility; b) planned analysis; c) multiverse analysis. Each branching represents a choice between different analysis options [43]. (Figure from Dragicevic et al. [20])

all alternative analysis paths are either not considered or deemed not suitable, for reasons not usually disclosed [20, 25].

One proposal to address the issue of undisclosed flexibility is *pre-registration* (see Figure 1b), where researchers commit to a single statistical analysis that has been planned and registered before collecting any data [11, 19, 45]. Although planning might partially address this issue, choices in a pre-planned analyses may still be arbitrary or have reasonable alternatives. Being able to specify alternative analyses in a principled way is therefore important. Wary of this undisclosed flexibility, some statisticians and methodologists have argued for approaches such as multiverse analysis, where analysts implement all analyses stemming from reasonable and justifiable choices in the data analysis process (Figure 1c). By surfacing all the possible decisions that go into data construction [55, 56] and model building [47, 55], and by reporting the outcomes from these myriad analyses, such approaches promise an increased transparency as well as greater understanding of the sensitivity of outcomes on analytical choices. This allows the reader to quickly understand and appreciate the robustness or fragility of the result to arbitrary, yet defensible, decisions that researchers might make during an analysis.

In describing the steps that go into creating a multiverse analysis, we adopt the "tree of analysis" metaphor (Figure 1): "an analysis proceeds from top to bottom, and each branching represents a choice between different analysis options" [20]. Borrowing terminology from Dragicevic et al. [20], an *analysis parameter* represents a node in the tree that has more than one child—a point in the analysis where the analyst must decide between reasonable alternatives—and an *analysis option* is one of those children. A singular *analysis* (i.e. universe) is a complete path from root to leaf. Although not exhaustive, Dragicevic et al. [20] identified four types of *analysis parameters* corresponding to different types of decisions that are likely to come up in data analysis: data substitution, data processing, modeling and presentation; and types of *analysis options* that may have been identified for each *analysis parameter*.

For the multiverse library to be sufficiently expressive, we identify the need to **flexibly support each type of *analysis parameter*** as our first design goal (D1). Moreover, when implementing an analysis parameter in code, an analyst may need to change their code in more than one place (e.g., a decision may impact data transformation and a model at the same time). Such changes are a consequence of the same decision, and thus should be represented by the same *analysis parameter* syntactically.

In addition, in a multiverse analysis certain choices made for one parameter may be inconsistent with choices for other parameters,

resulting in inter-dependencies across multiple parameters [56], which are referred to as *procedural dependencies* [37]. To ensure expressiveness of the tool, `multiverse` should **support specification of procedural dependencies** (D2).

2.2 Authoring Multiverse Analyses

To create a multiverse analysis, the analyst must generate and execute all unique combinations of compatible analysis options, at each analysis step, resulting in a large set of unique end-to-end analyses for the same dataset. Despite a growing number of studies using multiverse analysis or similar approaches (e.g., [2, 7, 8, 10, 12–15, 18, 46–48, 55–57]), writing code to execute all reasonable analysis paths (excluding unreasonable paths) in a multiverse can be quite cumbersome, and few tools exist to aid this process.

Some tools, such as Mrobust [62] and SpecR [41], are specifically designed to support multiplexing over model specifications. These allow users to substitute variables in a statistical model, but leave out other important parameter types such as data processing and data substitution [19]. Other tools, such as rdfanalysis [22] and Boba [37], allow analysts to multiplex over the full space of analysis parameter types, but come with other constraints on the user's workflow. rdfanalysis requires users to adopt a specific, rigid format for defining analysis steps and alternative analyses, where decision points and their corresponding code are defined as objects across separate template files. Boba provides a flexible syntax and makes it easier to build up a multiverse from a single analysis, but does not support the iterative workflow of a computational notebook. We draw inspiration from these prior efforts, and push further in the direction of integrating our tool into existing computational notebook environments to support interactive EDA.

2.3 Data Analysis Workflow

Data analysts have been characterised as engaging in an iterative and exploratory process which provides "flexibility, discovery, and innovation" [40]. This bears similarities with end-user programming [31], a style of programming that is referred to as *exploratory programming* [52].

Literate programming tools, which allow clear communication of analysis and code interleaved with one another [35], have also become increasingly popular (e.g. RMarkdown, the computational notebook for R, and Jupyter Notebooks), and are taught to students in introductory data science courses as a part of the data science workflow [36]. Such tools can be particularly useful during EDA, as they provide users with in-context, timely feedback, making it easier to iterate on ideas quickly and diagnose errors.

While engaged in exploratory programming, analysts often need to "make a cost/benefit tradeoff between producing high-quality code, and spending their time and effort on quick ideation" [32]. This results in code which is "messy", "ad hoc", "experimental" and "throw away" [27, 30, 34, 49], and scripts and notebooks with old code are kept around for provenance tracking [33]. In many ways, keeping track of duplicates and history is analogous to implementing alternative analysis paths in a multiverse. Tools which support local versioning of code for provenance tracking and managing exploratory code [31, 33] and prototype designs [26] can help inform the design of tools to support authoring multiverse analysis.

Taking inspiration from both the literate programming and versioning approaches, we identify three additional design goals for `multiverse`. First, we need to **support existing workflows of data science workers (D3)**, by making multiverse authoring compatible with computational notebooks and conducive to in-context, interactive exploratory data analysis. Second, to limit the disruption to existing workflows, we need to ensure **closeness in syntax to the dominant paradigms of the underlying programming language (D4)**; R, for example, is a functional programming language. The syntax for a multiverse analysis authoring tool built on R should bear resemblance to its syntax, and the implementation should provide sufficient abstraction to be similar to regular R use. Finally, we should **support analysts in declaring alternatives through local changes in the code (D5)**; this will help retain the context of the surrounding code in which decisions in the multiverse specification are being made, without affecting the shared aspects of the analysis, facilitating iteration.

3 DESIGN AND IMPLEMENTATION OF `MULTIVERSE`

Building on design requirements (**D1-D5**) summarised in Table 1, we developed `multiverse`, a tool designed to enable analysts to implement multiverse analyses. `multiverse` fully integrates within state-of-the-art RMarkdown computational notebooks (**D3**), but our general approach could be extended to other formats.

To specify a multiverse, an analyst must provide code corresponding to the different *analysis options* available at each step in the analysis. The `multiverse` package uses the meta-programming capabilities of R to allow analysts to replace any sub-expression in their R code with a `branch()` statement, the core operator of the package. The `branch()` statement simultaneously defines a new *analysis parameter*, its corresponding *analysis options*, and the code that should be inserted at that point in the analysis for each *analysis option*. This allows analysts to succinctly and quickly build up complex multiverses from elementary decisions at any point in their code. The `multiverse` package then produces and executes all *analysis paths* through the code, corresponding to all valid combinations of *analysis options* (subject to any *procedural dependencies*).

We outline how a user can progressively build a multiverse analysis using our tool, and how our design requirements (**D1-D5**) are supported. We focus on the data collected by Jung et al. [29]

D1	flexibly support each type of analysis parameter
D2	support specification of procedural dependencies
D3	support existing workflows of data science workers
D4	closeness in syntax to the dominant paradigms of the underlying programming language
D5	analysts in declaring alternatives through local changes in the code

Table 1: A summary of the design goals for `multiverse` that were presented in §2

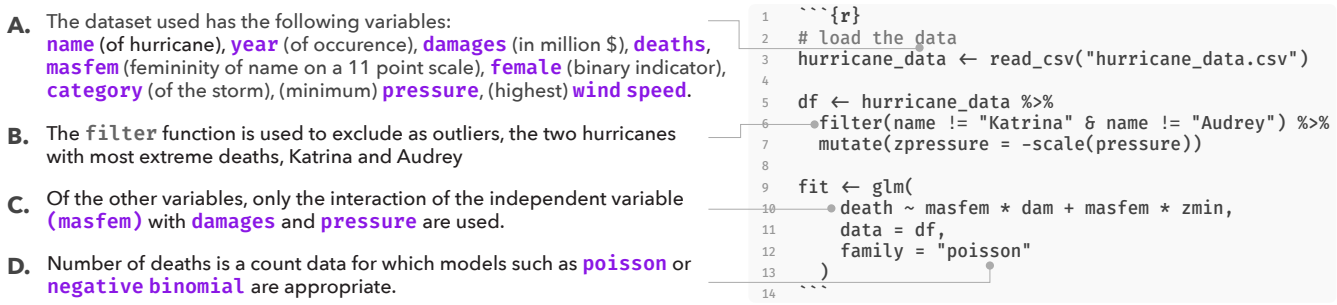

Figure 2: An implementation of the original (single-universe) analysis conducted by Jung et al. [29] in R.

investigating whether hurricanes with more feminine names lead to more deaths than those with more masculine names. Based on several critiques of the original study [3, 9, 38, 39], Simonsohn et al. [55] created a multiverse analysis of this dataset. We recreate this multiverse analysis using `multiverse` to demonstrate the different functionalities our tool supports. We also describe how `multiverse` can be used for script-style programming.

3.1 Data Collection and Original Analysis

The dataset used by Jung et al. [29] contained information on 94 hurricanes from a list published by National Oceanic and Atmospheric Administration (NOAA). For each storm, the authors compiled information on the year (`year`),[2] number of deaths (`deaths`), minimum pressure (`pressure`), maximum wind speed at time of landfall (`wind`), dollar amount of property damages (`damage`) and hurricane severity or category of the storm (`category`). Nine independent coders were asked to rate the names of the hurricanes on a two-item 11-point scale (1 = more masculine; 11 = more feminine), and the `femininity` of each name was computed as the mean of these two items.

Figure 2 outlines the steps involved in implementing Jung et al.'s analysis [29], in which the two hurricanes with the highest death toll were removed as outliers (Figure 2B). To test their hypothesis, the authors fitted a negative binomial model using the number of deaths as the response variable. For predictors, they used `femininity`, `damages`, standardised value of pressure `zpressure`, interaction between `femininity` and `damages`, and the interaction between `femininity` and `zpressure`.

Several subsequent studies [3, 9, 38, 39], each proposing a different analysis strategy, found inconclusive or opposite results, suggesting that the original finding may have been the result of an idiosyncratic combination of analysis choices. Simonsohn et al. [55] summarised all of these alternative approaches, and identified a number of different data transformations and modeling choices for this dataset, producing a multiverse analysis.[3]

[2]the name of the variable in the dataset is given within the parentheses
[3]Simonsohn et al. [55] use the term specification curve analysis to refer to a multiverse analysis

3.2 Specifying a Multiverse

We describe how an analyst might progressively create a multiverse from the bottom up[4], by modifying the single-universe analysis of Figure 2 of the hurricane dataset.

3.2.1 The multiverse object. To start, we must first create a new `multiverse` object: `M <- multiverse()`. The `multiverse` object serves as the interface for interacting with the multiverse analysis that is being created.

3.2.2 multiverse *code chunks.* Computational notebooks interleave rich text used to describe an analysis with *code chunks* implementing the analysis itself. In RMarkdown, R code chunks are indicated by chunk delimiters: ```` ```{r} ... ``` ````. Creating a multiverse code chunk (**D3**, **D4**) is similar: ```` ```{multiverse} ... ``` ````. Unlike regular R code chunks, multiverse code chunks require two arguments: the *label*, a unique identifier for the code chunk, and *inside*, the multiverse object. Labels are optional in R code chunks; `multiverse` makes them mandatory. This allows the tool to uniquely identify code chunks as users modify them interactively, which is necessary to internally keep track of modifications to code chunks in the multiverse object. In the RStudio IDE, we provide a shortcut to create code chunks with unique names, which users are free to change.

To use functionality from the multiverse package in our hurricane analysis code, we change the code chunk type from `r` (Figure 2,

[4]While our example features a particular sequence for creating a multiverse, this order is not necessarily prescribed, and could have been different.

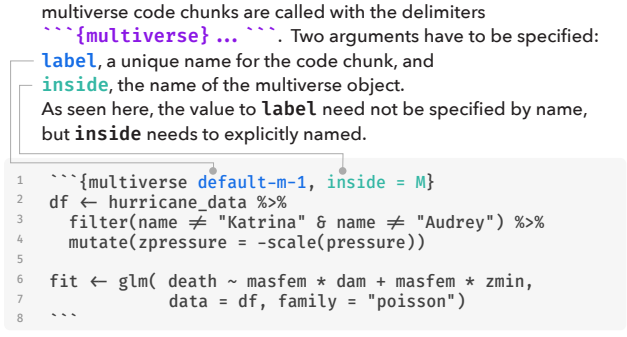

Figure 3: To create a multiverse analysis in RMarkdown, users can use a dedicated code chunk which will make calls to multiverse compiler. It can also be used as a regular R code chunk.

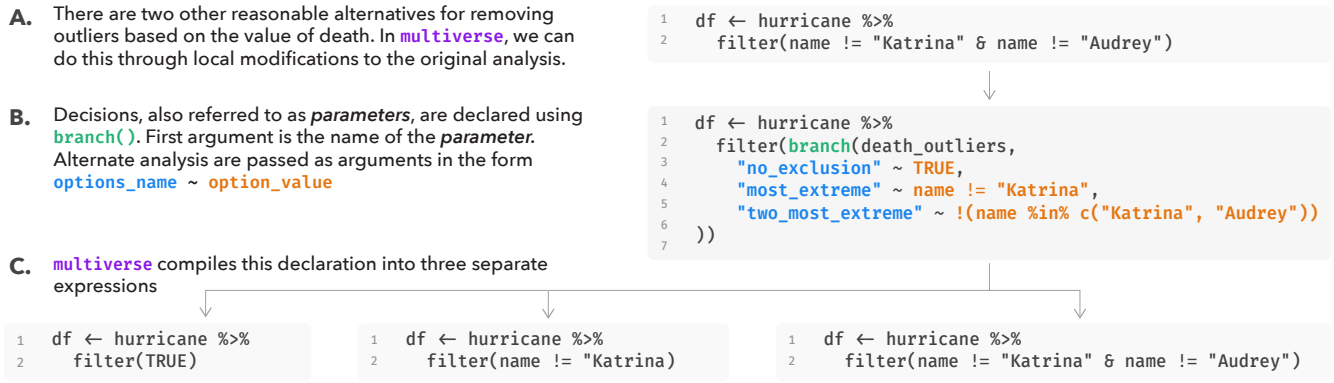

Figure 4: branch allows declarations of alternative ways of performing an analysis step which is multiplexed over, to create the multiverse

line 1) to `multiverse` (Figure 3, line 1). Because we have not yet specified any branches in our code, the output of the analysis remains unchanged, but we can now use `branch()` statements.

3.2.3 `branch()`*: creating an alternative analysis path.* The first decision point in this analysis is to determine which points to exclude as outliers. The original analysis removes the two hurricanes causing the highest number of deaths to improve model fit, as including these two data points resulted in over-dispersion (Figure 2B). In considering alternative ways to define outliers, Simonsohn et al. developed a principled exclusion criteria based on deaths and damages. These criteria become progressively more strict, resulting in three ways to exclude outliers based on deaths and four ways based on damages, yielding twelve combinations of ways to exclude outliers.

We first implement the three possible exclusion criteria based on deaths. We create an *analysis parameter* called `death_outliers`, with three possible *analysis options*: `no_exclusion`, `most_extreme`, and `two_most_extreme`. We replace the Boolean condition in the `filter` call (Figure 2, line 6) with a `branch()` statement defining each *analysis option* and the alternative code that should be inserted for each option (Figure 4B lines 3-6). `branch()` can be used to replace any subexpression of R code with a set of alternatives, in a flexible manner (**D1**, **D5**). In this example, we take advantage of this property of `branch()` to replace just the boolean subexpression that determines which rows of the dataset to keep; this allows us to succinctly define different exclusion criteria. This approach also fits well into the functional programming style familiar to R users, as all statements in R are functions and can be recombined in similarly flexible ways (**D4**). As this is the only `branch()` statement so far, our code currently defines three unique *analysis paths*: one for each *analysis option* in the `death_outliers` parameter (Figure 4C).

We can similarly create a `damage_outliers` *analysis parameter* defining four additional ways to exclude outliers based on damage (Figure 5). Below we describe how the `multiverse` package creates and executes all $3 \times 4 = 12$ analysis paths represented by the combinations of these two parameters' analysis options.

3.2.4 Reusing analysis parameters in `branch()`. Jung et al. [29] fitted a model with `deaths`, as the response variable. `deaths` is a count variable with a long-tailed distribution, for which Jung et al. decide a negative binomial model would be appropriate (see Figure 2D). Although a reasonable choice for count data, one can also reasonably argue for a log-linear regression model instead. In contrast to the previous example of exclusion criteria, specifying the model type requires changing *two* different locations in the code: the specification of the `deaths` dependent variable (Figure 2, line 10) and the value of the `family` argument (Figure 2, line 12). However, these are not two separate decisions (i.e. *analysis parameters*), but rather a consequence of the same *analysis parameter*: the choice of model.

Often, a single analysis parameter will require the analyst to change the code in more than one location. To represent these semantics, `multiverse` allows us to re-use the same *analysis parameter* in multiple `branch()` statements (**D1**), so long as each `branch()` uses the exact same set of *analysis options*. In a branch on a previously-defined parameter, option names must be the same, but the code for each option can be different. Thus, we represent the consequences of the choice of model with a single *analysis parameter*: `model`. We insert two `branch()` statements using this parameter, one to set the variable transformation (Figure 6, lines 2–4) and one to set the `family` (Figure 6, lines 14–16).

3.2.5 Specifying procedural dependencies with `%when%`. Another decision made by Jung et al. [29] was their choice of predictors (Figure 2C), which includes the interaction between `femininity` and `damage`, and between `femininity` and `zpressure`. Here, the predictors `damage` and `zpressure` are used as measures of the storm severity, with the interaction between `femininity` and `damage` indicating whether the main effect is stronger in more severe storms.

```
df ← filter(df, branch(damage_outliers,
  "no_exclusion" ~ TRUE,
  "most_extreme" ~ name != "Sandy",
  ...<2 more analysis options>
))
```

Figure 5: Subsequent branch calls will progressively expand the multiverse, by enumerating all possible combinations

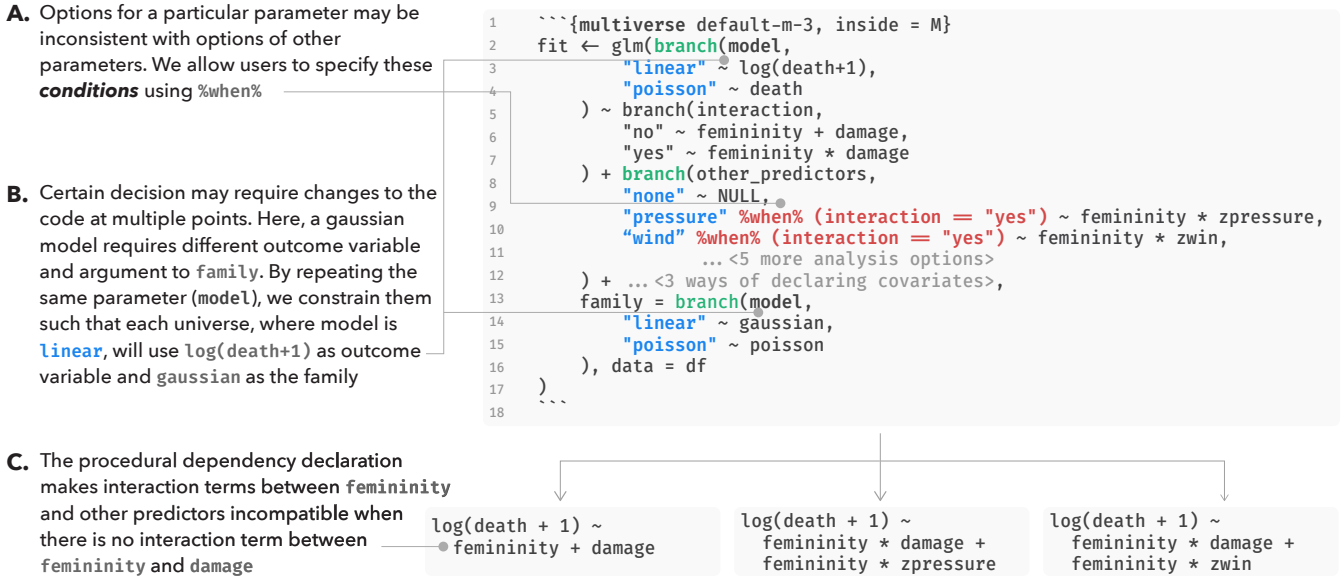

Figure 6: User can define procedural dependencies between options with the %when% operator. Users can also reuse parameter names for decisions which manifest in more than one location.

There are other reasonable approaches to study the primary effect which may include no interaction term or interaction with other variables such as `category` or `wind`. In this case, Simonsohn et al. [55] only included interactions between `femininity` and variables such as `pressure`, `wind` or `category` in conjunction with the interaction between `femininity` and `damage` (Figure 6A, line 10–11).

In a multiverse analysis, there may arise such *procedural dependencies* between two or more *analysis parameters* that make certain analysis paths inconsistent, or impossible. In other words, the applicability of some *analysis options* may be conditional on an upstream decision. By default, `multiverse` assumes all combinations of options are valid. However, it provides a flexible way to specify that an *analysis option* is incompatible with previously-specified *analysis options*. These *procedural dependencies* can be specified using the `%when%` operator followed by a Boolean expression, after the *option name* (**D2**). We replicate their analysis—using the `%when%` we make sure there are no interaction terms between other predictors in the absence of the interaction term between `femininity` and `damage`.

3.3 Executing Multiverse Code

3.3.1 Executing a code chunk interactively. As with code chunks in a typical computational notebook, users can execute a multiverse code chunk in the interactive editor in RStudio. When a user executes a single code chunk, `multiverse` internally transforms the input from that code chunk into one unevaluated expression (R's internal representation of an abstract syntax tree) for each unique combination of *analysis options* (§3.7) and immediately executes the *default analysis*: the *analysis path* obtained by taking the first *analysis option* at each decision point. The output of an executed code chunk is displayed immediately below it (Figure 7), mimicking notebooks. Analysts can change which analysis path is executed by default. Inline code output of a single analysis path (and the ability to select that path) is meant to support the familiar trial and error workflow of data analysts and to aid debugging.

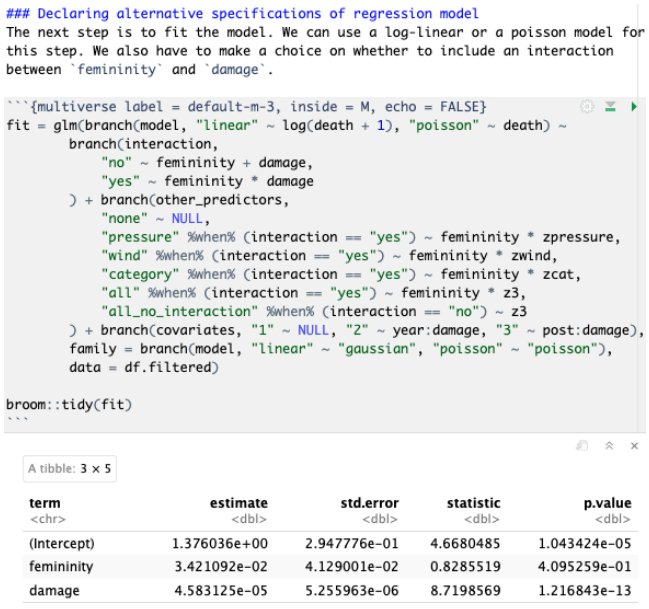

Figure 7: Use of `multiverse` code chunks in R Markdown. Upon executing, code chunk, the user gets the results of the default analysis. Here, the default analysis fits a log-linear model and prints the coefficients of the model as a dataframe.

3.3.2 Executing the entire multiverse. To execute all unique *analysis paths* in the multiverse, an analyst can call `execute_multiverse()`

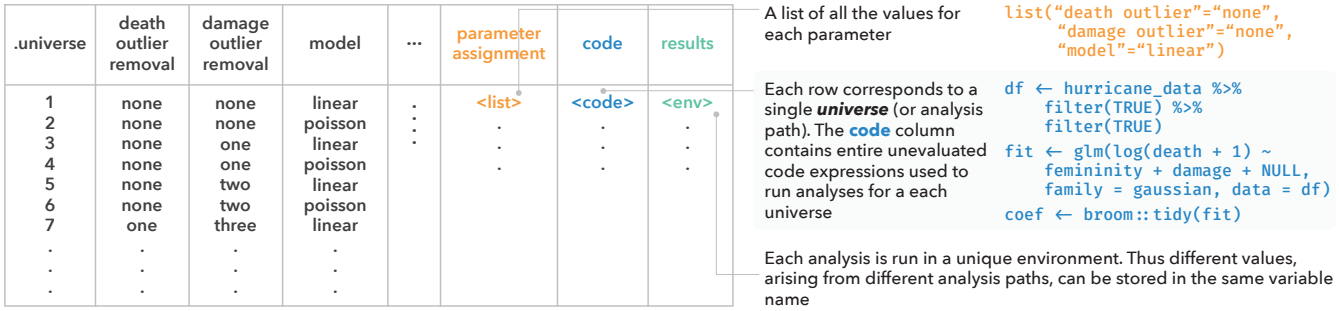

Figure 8: `expand()` provides an overview of the complete decision tree of the specified multiverse, with each row corresponding to the set of decisions creating the a particular analysis path.

on the multiverse object. `multiverse` supports executing the individual analysis across multiple cores or computing clusters using existing parallel computing packages in R, such as `future` [4].

3.3.3 Inspecting the result. To help keep track of the declared analysis paths and, on execution, inspect the results from each path, an analyst can call `expand(M)`. This will return a table where each row corresponds to a single analysis path i.e., a single *universe* (Figure 8). This view provides the user with the information of which choices have resulted in the analysis path, the entire unevaluated code expression corresponding to each analysis, and an environment which stores the result of the analysis corresponding to the *universe*. Analysts can use this table to explore multiverse specifications with all the tools available in R and RStudio for exploring data tables. Analysts can use `extract_variables(M, <variable names>)` to extract the supplied variable from the results of each analysis path, returning a table similar to the output of `expand(M)`, but with new columns for each variable that has been extracted. This would allow an analyst to perform tasks such as investigate the result or extract summary statistics or data tables from all universes simultaneously.

3.4 Debugging and Diagnosing Errors

During execution of the default analysis, `multiverse` provides the same set of debugging utilities that R provides. When the user executes the entire multiverse, the tool outputs the error message, a traceback—an object containing the entire call stack that caused the error—and the index of the corresponding analysis path in which the error was encountered. The execution of the remaining analysis paths in the multiverse are not halted if any errors are encountered. The traceback is helpful to identify the location of the error, as often R expressions return unidentifiable error messages. `execute_universe(<universe ID>)` (universe ID are found in the table output by `expand()`, see Figure 8) allows analysts to execute a particular analysis path and reproduce errors encountered in the execution of that specific path.

3.5 Additional utilities

In a multiverse, the number of universes can grow extremely quickly, making it challenging to keep track of all possible analysis paths. In our simplified example, we defined five *analysis parameters*—`death_outliers`, `damage_outliers`, `model`, `main_effect`, `other_predictors`, and `covariates`—with two to six *analysis options* each. Accounting for our procedural constraints, there are 504 unique analysis paths in this multiverse. Simonsohn et al. [55] identified 1728 unique paths in their analysis. Keeping track of so many analysis paths can entail significant cognitive load. We provide functions to help users get an overview of the multiverse specification, as well as perform exploratory data analysis using the extensive set of tools and utilities that R itself provides, such as ggplot2 [58].

For instance, `multiverse` provides the `extract_variables` functions to extract results from the multiverse analysis as additional column(s) of the table described in Figure 8. Analysts who want to better understand the composition of the multiverse can incorporate this data structure into an analysis pipeline with other R libraries and create their own visualisations. The `multiverse` library contains documentation which describes how to create commonly-used visualisations such as specification curves [55]. We describe these functions, and how to invoke them, in more detail in the supplement.

3.6 Using `multiverse` in R Scripts

`multiverse` also supports users who prefer traditional script-style programming over computational notebooks. In a script, users can create a multiverse analysis by declaring multiverse expressions with `inside(M, { ... })`, which can then be executed or manipulated using the same functions described above, e.g. `execute_multiverse`. Thus, the multiverse declaration from Figure 4 would become:

```
inside(M, {
  df <- hurricane %>%
    filter(branch(death_outliers,
      "no_exclusion" ~ TRUE,
      "most_extreme" ~ name != "Katrina",
      "two_most_extreme" ~ !(name %in% c("Katrina", "Audrey"))
  ))
```

3.7 The Multiverse Compiler

When users execute a multiverse code block or a call to `inside()`, the `multiverse` compiler takes the declared R code with `branch` statements and transforms it into multiple unique R expressions, one for every possible analysis path. This is done in two steps. First, `multiverse` enumerates all possible valid *parameter assignments*.

A *parameter assignment* consists of a single assignment of an *analysis option* to each *analysis parameter*. Valid parameter assignments are those that satisfy all procedural dependencies (e.g., `%when%` clauses). Each parameter assignment corresponds to a single possible analysis path through the multiverse; or a single *universe*. Second, `multiverse` takes each *universe* and uses its *parameter assignment* to recursively rewrite the unevaluated multiverse code expression into code specifically for that universe. It does this by replacing each `branch()` subexpression with the subexpression corresponding to the parameter assignment for that branch.

3.7.1 Optimisation. Consider a multiverse with *m analysis parameters*, and *n analysis options* per *analysis parameter* on average; this results in a multiverse with n^m distinct analyses. The naive approach of executing all *analysis paths* separately would result in an execution time of ($O(mn^m)$). We use a tree structure to store the output of the multiverse compiler, with each level in the tree corresponding to a single *unit* of the multiverse. If the analyst is using multiverse code chunks, each *unit* is a single code chunk; if the analyst is using `inside()`, each *unit* is one call to `inside()`. When the user declares a *unit* with one or more *analysis parameters*, we enumerate all combinations of the *analysis options* which are added to each existing node from the previous level as children, followed by pruning any inconsistent analysis paths declared as *procedural dependencies*. For the first *unit*, all nodes are the children of a singular root node. Thus, if we declare a multiverse with *m analysis parameters*, 1 *analysis parameter* per *unit* on average, and *n* analysis options per *analysis parameter* on average, we have a multiverse with n^m distinct analyses and a tree with approximately $\frac{(n^{m+1}-n)}{(n-1)}$ nodes. By executing just the code at each node in the tree, we reduce the execution time to: $O\left(\frac{(n^{m+1}-n)}{(n-1)}\right) = O(n^m)$.

This is possible because in R, environments are mutable dictionaries that encapsulate variable state and power scoping in the language [59]; all code in R is executed inside an environment that determines the variables defined for that code and the values assigned to those variables. Moreover, every environment has a parent environment, and variable bindings in the parent environment are accessible to its child environments. We execute each node in its own environment. Because we share parent environments for nodes at the same level of the tree, two nodes which are children of the same parent node will automatically have results of the parent node's computation available in their execution environment, avoiding redundant computation.

3.8 Refinement from Early Feedback with Users

To arrive at the design of `multiverse` described above, we first built a prototype version based on our initial design criteria, then conducted a series of informal semi-structured formative evaluations[5]. The goal of these sessions was to assess if our approach fit into the computational notebook workflow and to uncover early usability issues with the tool. We recruited six graduate students or faculty in HCI and Psychology, who were all experienced R users. These sessions allowed us to refine the syntax of the package by identifying difficult-to-use or redundant aspects of the API.

Our early interviews motivated the need for the specialised code chunk syntax. Our initial implementation provided the `inside()` function and allowed users to provide code defining variables in the multiverse using R's operator for accessing elements of an object (the $ operator). We found that the syntax with $ was confusing to users, and that neither syntax supported as fluid iteration in a notebook as we had envisioned—they introduced extra boilerplate, and did not as easily support directly running code chunks or seeing output from a single universe. This led us to implement the `multiverse` code chunk approach. We kept the `inside()` function for use with script-style programming, and removed the $-based syntax altogether.

4 EVALUATING THE DESIGN SPACE OF MULTIVERSE PROGRAMMING INTERFACES

We performed a comparative analysis with two other programming interfaces that support creation of multiverse analyses. We implemented the same multiverse analysis of the hurricane dataset using each of these tools to highlight the different design choices in each of these tools. We provide the specifications in our supplement, and below we describe relevant differences in the syntax and semantics of each package. Specifying the same multiverse analysis in each tool helps to ground our evaluation of all three packages, including `multiverse`.

4.1 Other Multiverse Programming Interfaces

4.1.1 mverse. mverse [42] was developed as an extension to `multiverse` specifically designed for educational purposes. The goal of mverse is to provide high-level abstractions for declaring alternative analysis paths to make it easier for novices in R and programming in general to specify a multiverse. mverse does not provide a universal function for declaring local variations in analysis paths, but instead provides distinct functions for multiplexing as analogs of existing functions in R. In part, mverse is an exercise in pushing **D4** (closeness in syntax to the dominant paradigms of programming in R) to its extreme, allowing us to better understand tradeoffs in the design space of multiverse tools. For example, choosing how to exclude outliers can be implemented in mverse using the dedicated `add_filter_branch()` function. Due to these design choices, data wrangling requires lighter, less complex syntax that is stylistically closer to the `tidyverse` grammar—a set of R packages which "share an underlying design philosophy, grammar, and data structures" [1]. While this simplification might improve learnability and reduce errors for novices, it requires mverse to define separate methods for declaring, adding and removing different types of transformations, at the cost of **D1** (flexibly support each type of analysis parameter).

4.1.2 Boba. Boba [37] is language-agnostic, and users create multiverse analyses in a template file, which can be created using a text editor or an IDE such as RStudio. After specifying their analysis, through a command-line tool, users can compile their analysis into R Scripts (one script for each analysis), execute the scripts, and merge the results from the scripts together The template file

[5] see Supplementary Material for the interview guide

consists of two separate parts: (1) a Boba config block containing JSON, and (2) an implementation of the analysis (e.g. R or Python). This separation of the analysis is similar to C++-like separation of header and implementation, and to describe it, we make a distinction between specifying the decision space *locally* at the actual site of each analysis parameter in code versus *globally* in a single block of JSON.

Boba supports three ways to create analysis parameters and options—by specifying "decisions" as JSON globally in the Boba config block, by writing options locally in declarations, or by using the "code block" syntax for options that span multiple lines of code. All code in the template after the final option written in "code block" syntax is parsed by Boba's compiler as a part of that final option. In order to write code which is common to all analysis paths following the use of "code block" syntax, the user has to declare an additional code block which consists of a single option to indicate the end of the blocks associated with the previous analysis parameter. This adds a parameter to the decision tree which does not actually represent a conceptual analysis decision. To specify procedural dependencies in Boba, users can either write "constraints" globally in the config block or use `@if` operator to declare them locally. Users can also indicate that the same conceptual decision manifests at multiple locations by using a "link" constraint in the config block.

Table 2 summarises the high-level differences between the tools.

4.2 Summary of Evaluation Using Cognitive Dimensions of Notation

We draw on existing evaluative frameworks in HCI to compare the design of `multiverse`, Boba, and mverse programming interfaces.[6]

[6]We initially intended to include rdfanalysis [22] for this discussion, but our implementation and evaluation revealed that most of the issues with this library stem from the rigid, heavily templated coding style that it imposes on the user; hence we excluded it from our analysis.

The *cognitive dimensions of notations* [6, 24, 32] break down representational systems (e.g., programming libraries) into orthogonal considerations about how well they support user reasoning; and the *gulfs of execution and evaluation* [28, 44] address mismatches between a user's mental model and the system's affordances for expressing a user's intended analysis. Since our focus is on designing notations to support reasoning about and constructing multiverses, we concentrate our analysis on the API and environments for these tools. Three of the authors each independently evaluated the usability of the tools they were most familiar with. This qualitative coding involved systematically going through the cognitive dimensions of notations and noting anything relevant about a particular tool. The authors then met to discuss, compare, and synthesise notes. We report on our main findings below.

Progressive evaluation (*how the user checks work in progress*) and **Provisionality** (*level of premature commitment to actions*): Checking work in progress in a multiverse analysis involves executing analysis code, and the way that users do this is restricted by their computing environment. In `multiverse` and mverse, users can build up an analysis by adding decision points in a computational notebook and execute provisional parts of a data analysis which updates variables and data structures in the RStudio session. This supports iterative workflows [31, 32] (**D1**). In contrast, Boba requires users to author their analysis in a template file, then compile and execute from the command line. Evaluating work in progress requires the user to open a new session and run scripts representing individual universes, making it cumbersome to check provisional parts of code during the authoring process.

Consistency (*similarity of syntactic representations for semantically similar operations*) and **Closeness of mapping** (*how well notations represent the application domain*). Both `multiverse` and Boba conceptualise multiverse construction as consisting of two steps—a multiplexing step to declare alternatives, and a pruning step to declare incompatible combinations of analyses. `multiverse` provides

	Boba	multiverse	mverse
API style	Language-agnostic meta-programming (preprocessor)	Language-specific meta-programming (R)	Tidyverse-style R API
Parameter definition	global JSON: `"decisions" : { ... }`; local: `{{param = "o1", "o2"}}`; (local) code blocks: `#---(param) o1`	`branch()`	`*_branch()`, `add_*_branch()`
Condition definition	global JSON: `constraints: { ... }`; local: `@if`	`%when%`	`branch_condition()`, `add_branch_condition()`
Execution environments	Text editor or IDE, command-line and language specific environment	R and dedicated code chunks in RMarkdown	R or RMarkdown
Debugging utilities	Outputs messages, warnings and errors from each script to command line	Traceback objects for each universe where errors were encountered	Traceback objects for each universe where errors were encountered
Overview of decision tree	Prints table of partial decision tree; outputs CSV of decision tree on compilation	`expand()` creates a table of full decision tree which can be viewed in RStudio	`summary()` creates a table of full decision tree which can be viewed in RStudio

Table 2: An outline of the the design choices that each tool makes. Note that, as mverse is an extension of `multiverse`, there is overlap of certain design features.

a single core operator for each (`branch` and `%when%`), and its syntax stays close to existing syntax and conventions in base R. In Boba, the multiplexing step can be represented using two syntactic forms, JSON and code blocks (see §4.1.2); termination of a code block requires the user to declare an additional code block; and there is no support for the reuse of parameters for conceptually similar decisions. These raise potential issues of syntax consistency. Boba's syntax aims to represent decision spaces more broadly, regardless of programming language or execution environment and thus preserves closeness of mapping to the decision tree itself, which is expressed using a JSON structure. mverse conceptualises multiverse construction through analogs of familiar tidyverse and R functions, therefore preserving consistency with analogous functions.

Error proneness (*invitations for users to make mistakes or lack of protection against mistakes*) and **Hard mental operations** (*level of cognitive load*): Error proneness in each tool reflects unintended consequences of design choices that are otherwise well-motivated. The `%when%` syntax in `multiverse` impacts every instance of a given analysis option in a multiverse specification, not just code in the apparent scope of the condition; this may not be intuitive to all users. mverse users interact with wrapper functions (§ 4.1.1); while this makes the syntax less flexible and expressive, it should reduce error proneness because the operation of each function are specialised. Boba requires the user to write their template code in one editor but evaluates universe scripts in separate R environments, which requires the user to switch between different editing and execution environments—text editor, command line and R or RStudio IDE. This can involve greater cognitive load and create opportunities for errors.

Gulf of execution (*how difficult it is to express intended operations with a tool*): In `multiverse`, the `branch` operator allows users to replace any sub-expression in R to declare alternative analyses, but users have to determine *how* to multiplex over every type of operation they wish to employ. Similar challenges may be encountered in Boba, which uses text-substitution. mverse is limited by the existence of analogous functions for the task the user wishes to perform.

Gulf of evaluation (*difficulty interpreting whether a tool is behaving as a user intends*): Gulfs of evaluation arise when debugging or validating that a multiverse worked as intended. We expect a larger gulf of evaluation when this process is significantly different from the standard workflow of debugging individual paths. `multiverse` generates a tree structure of nested R environments which share their scope insofar as different universes share analysis code, a design choice meant to reduce runtime by eliminating redundant computations. This is different from usual program execution in R and thus different from the mental model of running code that an R user might have. Because of this execution process, debugging can be difficult. In contrast, Boba creates different execution environments for each individual universe script, executes them, and collates console logs and outputs the data from each universe. This involves processes that might be more familiar to a typical R user. As a consequence, errors are easier to reproduce by running universe scripts in an R session, making it easier to assess whether the implementation matches one's intention.

5 CASE STUDIES WITH RESEARCHERS

The `multiverse` library has been available for researchers to download from the CRAN R repository[7] for about a year. Several researchers discovered the library and used it to implement multiverse analyses. We report on case studies from three such researchers who have successfully implemented a multiverse-style analysis with the library, who we recruited for an interview. Each of them had different goals for implementing a multiverse analysis, and we describe what they consider a successful implementation based on the goals of their project. The case studies allowed us to gain insights into how the library is used in the real-world in varied scenarios, testing the breadth of the functionality of the API.

We conducted semi-structured interviews[8] over Zoom. We asked participants to first walk us through their multiverse implementation and observed how they had implemented their analysis. Based on our observations and our interview protocol, we asked them follow-up questions regarding the analysis. Interviewees were compensated $50 USD for a session of approximately 1 hour. Below we describe insights on participants experience and usage of the library.

5.1 Case Study: Creating a template for multiverse analyses

P1 is a graduate student in Psychology who has 5 years of experience with R, and is involved in a large-scale crowd-sourced replication project in psychological science, where they conduct multiverse analysis on over 70 open datasets. As the individual multiverse analyses were to be implemented by different teams of researchers, P1's goal was to develop an R code template which would be adapted by the other teams of researchers to implement multiverse analyses of their own on different datasets. P1 first identified various types of *researcher degrees of freedom* that are typical of psychology analyses, ordered and grouped them into categories (e.g. data transformations, outlier exclusion, etc), and compiled these decisions into a spreadsheet that acts as a framework for identifying and documenting *analysis parameters* and *options*. P1 then developed the R code template, which involved using the `multiverse` library to implement an example multiverse analysis.

P1 used the spreadsheet to create a conceptual outline of the multiverse analysis and identify all the *analysis parameters* and *options*. Prior to implementation, to familiarise themselves with the library, P1 copied code from the documentation and modified the examples for their data analysis. They then translated this conceptual analysis into R. They implemented the multiverse analysis in an R Script. It consisted of around 1200 distinct analysis paths. P1 stated that they found the library easy-to-use and that it was straightforward to translate their conceptual decisions into `multiverse` syntax. They developed a mental model for the `branch` function—*"I always thought in terms of* loops*; whenever I declared a branch statement, I thought of it as the multiverse package creating a loop"*.

While P1 was able to implement their intended analysis and develop a reusable template, they encountered certain challenges, specifically in implementing *procedural dependencies*—they found the Boolean statement declared with `%when%` difficult to evaluate

[7] https://cran.r-project.org/web/packages/
[8] see Supplementary Material for the interview guide

(*hard mental operations*), which in turn made it difficult to determine which combinations of *analysis options* were being rendered mutually incompatible. P1 also mentioned facing challenges in debugging and identifying the source of errors in their code (*gulf of evaluation*) due to the overwhelming number of error messages spouted by the library (§3.4). They adopted a custom workflow where they progressively added branches; if they encountered an error after a new branch was introduced, they copied a single analysis into a separate R script file and executed it to debug. Because P1 used an older version of the library, after the session we corresponded with them over email to get their feedback on the error messages they would have seen had they used the current version. They indicated that the error messages in the newer version of `multiverse` were more informative.

5.2 Case Study: Medical multiverse analysis

P2, a graduate student, first started using R six years ago but has used it more intensively in the last three years. P2 recently completed a study investigating the association between different levels of alcohol consumption and inflammation using a multiverse-style analysis. P2's analysis was motivated by mixed findings on the effects of different levels of alcohol consumption on inflammation, when compared to an abstinence reference group. Through the multiverse analysis, they found the association between low-to-moderate drinking and lower levels of inflammation marker to be robust to common variations in researcher-defined parameters, while the association between above-guidelines drinking and inflammation marker levels was less definitive.

P2 surveyed previous literature on the same research question for the different analyses that have been performed and created a table detailing the various degrees of freedom. Like P1, P2 also used `multiverse`'s documentation to get familiarised with the library. They then implemented their analysis in RMarkdown, adding *analysis parameters* where relevant. Their multiverse analysis consisted of approximately 1000 unique analysis paths, and was comprised of relatively simple types of analysis options, such as data processing, outlier removal, and modeling, which are fairly straightforward to define and are supported through many examples in the `multiverse` documentation. P2 considered including latent variable regression; however, they felt that such an analysis would be, in and of itself, complicated to define for a singular analysis. Thus, they omitted this analysis from their multiverse. We speculate that this might be an issue of cognitive load—a latent variable regression and multiverse analysis are both, independently, complex analyses and require *hard mental operations*; as such, P2 may have felt that including a latent variable regression within their multiverse analysis will compound implementation difficulty.

P2 experienced some challenges in defining *analysis options* for model formulas with different combinations of predictors (*gulf of execution*). As a workaround, they created strings for different groups of predictors. Then, inside a `branch` statement, P2 used text concatenation to combine those strings into the regression model formulas corresponding to each *analysis option*[9]. This is a complicated approach (*hard mental operations*) where the user first declares text and then constructs combinations of text, which are then parsed into R code. This is a consequence of the `multiverse` API's design, which substitutes parts of the abstract syntax tree with arbitrary sub-expressions—a conscious design choice to make `multiverse` syntax similar to R syntax (**D5**). Alternatively, a user could implement this without having to perform text concatenations, though this would require familiarity with the somewhat more complex metaprograming functionalities in R, such as quotation and quasi-quotation [60].

5.3 Case Study: Extremely large multiverse analysis in Virtual Reality

P3, a computer science graduate student, was an experienced R programmer having previously developed R libraries. P3 conducted a multiverse analysis to determine patterns in researcher degrees of freedom in how virtual reality users' motion is operationalised. A VR user's motion is determined based on 18 degrees of freedom from the position and orientation of VR headset and hand controllers. Through a multiverse analysis, they found that differences in measures of *synchrony*[10] were primarily due to how motion was defined (speed or velocity), transformation of variables (rank transform, log transform, or no transform) and the choice of the tracked point (head or hands). The analysis found that estimates of *synchrony* are sensitive to certain operationalisations, and surfaced the need for pre-registration of how users' motion will be defined. Like the others, P3 identified degrees of freedom primarily from prior work, which they mapped out on a whiteboard. The resultant multiverse was comprised of two million distinct analysis paths.

P3 used R Scripts and described finding the task of translating a conceptual decision to code with the library straightforward. P3's description of their implementation process was similar to the iterative workflow of data science workers. However, since they used R Scripts, they created a custom workflow where each conceptual decision was implemented in a separate script file and was tracked through a custom naming convention. Once they had implemented a few `branch`es, they felt confident in creating a single primary script.

P3's analysis was the most complex among our case studies. P3 needed to implement a nested decision which would have required a `branch` call within another `branch` call; or they would have to combine two decisions into a single *analysis parameter* which might cause issues downstream in their analysis. Instead, P3 used the `multiverse` API in a novel way—they combined two features of the API to create a new usage pattern that we had not explicitly designed—P3 created a parameter which has no code associated with it, and whose sole purpose was to specify the subsequent *procedural dependency*. This allowed P3 to succinctly specify the nested decision space:

```
branch(px_point_count, "one", "many")
px_data1.5 <- px_data %>%
  mutate(across(branch(px_body_parts,
         "all" ~ <...> %when% (px_point_count == "many"),
         "head" ~ <...> %when% (px_point_count == "one"),
         ...<other options>...
        ), .fn = ... ))
```

[9] see Supplementary Material for a detailed example of this implementation

[10] Synchrony refers to individuals' temporal coordination during social interactions. [17]

P3 liked the syntax for short declaration of *analysis options*, but since they had *analysis options* spanning 4-5 lines of code within some `branch` statements, they felt that the syntax got a bit messy in some cases. Finally, P3 experienced challenges in debugging and identifying errors, which was exacerbated by the extremely large multiverse they had defined. They found that certain debugging utilities in R, such as the stepwise debugger (`browser`) were not compatible with the `multiverse` API, and thus resorted to using `print` statements for debugging.

5.4 Summary of Findings from Case Studies

The case studies revealed that all three researchers were able to implement their desired multiverse analysis with the help of the API, found the `multiverse` library to be helpful to do so, enabling novel insights. The diversity of the projects and goals, and the range of backgrounds and programming expertise of the researchers, suggest that, overall, `multiverse` can support a wide range of analyses in a usable manner. Below we report on insights pertaining to **planning** analyses, **learning** curve, **expressivity** of the library, as well as **opportunities** for future developments.

We noticed that researchers engaged in extensive pre-planning before implementing their multiverse analysis, and degrees of freedom were primarily identified from prior work (**planning**). This may mean that, unlike typical data analysis workflows, researchers are less likely to brainstorm alternative analysis paths whilst implementing a multiverse analysis. However, researchers may still engage in an iterative workflow to refine and modify their code. To implement the conceptual multiverse using the API, researchers would start with the documentation that is provided with the library, and iterate on it until they were able to specify their desired analysis (**learning**). All researchers stated that translating the conceptual decisions to `multiverse` syntax was generally straightforward (**expressivity**). However, there were certain niche aspects of the API, such as the ability to reuse parameters (§3.2.4) which at least one user did not discover (**learning**), and developed custom workarounds to resolve (P3).

Interestingly, only P2 used RMarkdown notebooks for their multiverse analysis. P1 never uses RMarkdown, and P3 does use it but did not think it suited their analysis due to the size of the analysis. When they had to iterate on the specification or debug their code, P1 and P3 both adopted custom workflows, with P1's workflow being similar to our anticipated workflow using RMarkdown. Because P1 and P3 used R Scripts, they had limited support for *progressive evaluation* of their specified analysis. P2, on the other hand, used RMarkdown and multiverse code chunks for their analysis, and did not experience significant issues with debugging. This suggests a need to support *progressive evaluation* and *provisionality* for R Script users, which we discuss further in §6.2.

Our case studies also revealed how users may approach debugging when they encounter errors. We found that debugging utilities in R (e.g. the stepwise debugger `browser()`), did not interface well with `multiverse` (P3). This reveals an important opportunity for improvement. Additionally, P1's difficulties in declaring *procedural dependencies* with `multiverse` suggests a need to modify the syntax for declaring procedural dependencies to better reflect users' mental models. In addition, since a lot of *procedural dependency* declaration is not limited to the immediate scope but the entire specification, visualisations of the tree structure highlighting the pruned analyses may also make it easier for the user to reason about their declaration.

Finally, both P2 and P3 cited a need for further normative guidelines for multiverse analysis. P2 mentioned that most examples of visualising the results of a multiverse analysis, such as specification curves, are restricted to a single outcome of interest. However, it is not unreasonable to have an analysis where the researcher wishes to convey multiple parameters of interest. P3 was faced with the challenge of sense-making from an extremely large multiverse, which is computationally expensive, and wondered if there are appropriate sampling methods to identify which *analysis parameters* are relevant and which are not.

6 DISCUSSION

Our qualitative analysis of the design space for multiverse analysis programming interfaces reveals trade-offs in API design, and gaps in the design space where tools for debugging and conceptual reasoning about decisions have been underexplored. We point to opportunities and challenges to balance trade-offs in multiverse programming interface design, provide tools for debugging multiverse analyses, and help users reason conceptually about decision spaces.

6.1 Learnability versus Flexibility of APIs

There is a trade-off between the targeted and constrained multiplexing functions offered by mverse compared to more sweeping and flexible approaches that allow users to multiplex arbitrary code, as supported by `multiverse` and Boba. Imagine a data analyst that wishes to create a parameter representing different exclusion criteria: with mverse, the semantics of the verb map very closely to a single function, `filter_branch`. An analyst is likely to recognise this mapping and the difficulty is largely in determining the specific syntax of that function. In `multiverse` or Boba, an analyst must understand, in more depth, the semantics of separate operations—`filter`, an R function, and `branch`, a construct from the multiverse programming interface—to determine how to combine them to achieve the same result, as well as determining the valid syntax for that combination.

As mverse functions are analogs to existing API functions, it is likely to be easier for a user to develop mental models to perform these actions, but necessitates the existence of an analogous function in mverse for every task that the user might wish to perform, limiting expressiveness. Meanwhile, the design approach of tools like `multiverse`, which provides users with a universal statement to perform multiplexing (`branch`), might create a gulf of execution (§4.2). This problem is representatative of a core consideration in effective API design: what are the *atomic units* of an API, how do they recombine in larger conceptual units—high-level, abstract operations the user wishes to accomplish, such as *filter a dataframe based on a parameter*—and can users easily construct such combinations?

We are reminded of the large variation in visualisation APIs, from APIs that include pre-baked *chart types* like bar charts and pie charts, making them easier for a novice to pick up to APIs with coherent *grammars* for visualisation specification [50, 58], elegant

APIs with more general concepts of visual channels and geometries, which can be recombined into more complex charts—*if the user is able to determine how to do so.* Just like in other areas, there is likely no one multiverse API to rule them all; rather, a spectrum of API designs will be necessary to address different user groups' needs, carefully balancing flexibility and learnability. What these APIs should look like should be motivated both by formal elegance—we believe something like the `branch` statement is a good candidate for an atomic operation in more general multiverse authoring tools—but also empirical work on how people understand and construct multiverses.

Our case studies revealed that researchers are able to familiarise themselves with the syntax of `multiverse`, using the documentation as a reference. Following this initial learning phase, researchers seem able to use `branch` as an atomic unit to construct a complete multiverse analysis, as exhibited by three successful implementations, of which one consisted of over a million *universes.* P3's innovative usage pattern of `branch` to declare a nested decision lends further evidence that `branch` as an atomic operator can support flexible and expressive construction of multiverse analyses.

6.2 Diagnosing and Fixing Errors

The design of `multiverse` was based on the assumption that data analysts engage in an iterative workflow, and they predominantly use literate programming environments like computational notebooks for such iteration. However, our case studies revealed that users who prefer traditional scripting environments also engage in similarly iterative workflows, and have custom workarounds for the limitations of a scripting environment. To best support such iterative workflows in R Scripts, users should be able to declare *provisional* code and *progressively evaluate* their specified code, both of which would bring the usability of the `multiverse` API in R Scripts closer to the experience of users in RMarkdown. Currently, in R Scripts, users have to run the entire code specified within an `inside` function; adding functionality to allow users to execute parts of their multiverse analysis in the console without the `inside` wrapper could address the issue to an extent. In addition, improving debugging utilities that multiverse tools provide could be another way of supporting users.

However, a primary design challenge for debugging utilities in multiverse analysis is how to give the user detailed feedback without overwhelming them with information. Due to the combinatorial explosion of analysis paths in multiverse analysis, listing error messages for all analyses at once is not optimal; yet this is exactly what existing tools do. Boba provides the user command line output, including all messages, warnings, and errors encountered during execution along with the index of the corresponding universe. In `multiverse`, we provide tracebacks for each error encountered, which gives the user more relevant information for diagnosing errors. However, the resulting wall of error messages can still overwhelm the user (for e.g., P1 in our case study).

A more usable debugging tool might act like a traditional stepwise debugger, helping analysts quickly identify the exact analysis path and location where their code failed, as well as load variables from the relevant analysis path into the current environment in the user's IDE. In a notebook environment, the ideal debugger may even load in a full notebook environment and allow the user to probe and execute code chunks directly. This would enable the user to immediately begin probing for the reason why their code failed without having to manually navigate to and re-execute the analysis path that produced an error. It would also bring the output of error messages in line with the way that users debug these errors: one-at-a-time.

Realtime visualisation of the structure of the multiverse could aid in debugging as well. A visualisation that continuously updates as the analyst creates branches might reveal hidden dependencies before the analyst even executes the multiverse. Such an approach could prevent errors before they occur, rather than requiring analysts to discover hidden dependencies through debugging tasks, which are cognitively demanding even in traditional (non-multiverse) programming. Interactive overview visualisations of the decision tree could also be used when executing the multiverse, showing where errors and warnings are occurring in which analysis paths in realtime. This might enable the user to investigate errors or prune the decision tree mid-execution. This would spare the user time waiting for universes where models are struggling/failing to converge. A persistent visualisation of the decision tree could also be an interface to the environments for each universe, such that by clicking a particular leaf node the user could be dropped into a particular universe. We imagine these visualisation tools as fully integrated with a notebook authoring environment like RStudio, rather than separate tools that require analysts to adopt completely new workflows.

6.3 Dealing with Large Multiverses

A cause for concern while implementing multiverse-style methods is the possibility that an analysis requires a very large multiverse, with thousands or even millions of distinct specifications. In such scenarios, researchers must apply care to ensure that, for each decision, every alternative option is equally reasonable and justifiable, and the decision to choose between them is entirely "arbitrary" [55]. If a multiverse specification includes options which are "unambiguously inferior", the analyst risks "drowning out reasonable effects in a sea of unjustified alternatives" [16]. Thus, if theory or relevant background knowledge provides sufficient indication that an option is inferior to the other options, then the corresponding specifications should be removed from the multiverse analysis. Del Guidice and Gangestad [16] illustrate how two multiverses, a large one where some specifications may be inferior and a smaller one where all specifications are equally justifiable, can lead to different conclusions. In the larger multiverse, the analysis revealed a wide range of both positive and negative effect sizes, which may lead one to interpret the effect of interest as not robust. In the smaller multiverse, the results indicate two clusters of effects, due to uncertainty around the causal nature of the effects being studied. This example highlights the need for analysts to not approach multiverse analysis as a specification maximising problem, but instead place emphasis on ensuring that all specifications are equally justifiable.

Unfortunately, by providing syntax to specify decision spaces and operationalising the multiverse as a cross-product of decision options, existing tools (including `multiverse`) may promote multiverses which are probably too large and which contain unreliable

estimates. However, certain features of existing tools also hold promise for pruning out analysis options that are non-equivalent in principle *a priori*. This requires the user to recognise during the specification process that specific analysis options do not make sense to include, either because they are less justifiable than alternatives or because they change the meaning of the final result [16]. rdfanalysis [22] elicits the rationale behind decision options by requiring users to document their choices. This kind of elicitation might integrate better with existing literate programming workflows [35], for example, by using prompts inside of template notebooks to scaffold the user's reasoning [61]. Boba's web-based Visualiser enables the user to perform exploratory data analysis on multiverse results, providing specialised views for evaluating model fit, inferential uncertainty, and robustness of results to analysis decisions [37]. Although these visualisations do an excellent job of revealing which analysis paths are non-equivalent in principle, ideally users should do as much pruning of the decision space as possible during the specification process. Future work could explore ways of integrating visualisations like those in Boba for checking model fit into a computational notebook environment where these tools could be used to evaluate and iterate on provisional analysis options in context.

However, it is entirely possible that despite considering only equally reasonable specifications, one ends up with a multiverse analysis with millions of distinct, equally justifiable analysis paths (for example, P3). For such large multiverses, keeping track of the declared specifications, and making sense of the entire analysis can be challenging as existing tools provide limited support for these tasks. While visualisations such as specification curves can aid in sense-making to a certain degree, there is a potential for dedicated interactive visualisation interfaces to assist the analyst in such tasks [25]. We hope to address this in future work.

Moreover, running each of these analysis paths may be extremely computationally expensive. A potential solution to get around the problem of analysing very large multiverses may be to sample a subset of the declared specifications in a principled manner, analyse and interpret them. However, applied researchers currently lack guidance and formal theory on how to effectively sample from the large space of distinct analyses. This points to a broader issue that researchers face—as multiverse analyses grow more popular, there needs to be greater support on how to implement and interpret results from a multiverse in a principled manner, akin to developments in Bayesian model development [21, 51].

6.4 Lowering Barriers for Transparent Statistical Reporting

`multiverse` is designed specifically to work within the RMarkdown authoring environment in RStudio. This decision allows the tool to make use of and extend the report authoring utilities provided by these platforms, getting us closer to realising the idea of *exploratory multiverse analysis reports* [20]: self-contained, interactive papers that allow readers to interactively explore the results of a multiverse analysis. Our work does not attempt to address larger incentives among researchers to publish strong results, which may also impact researchers' use of approaches like multiverse analysis. However, if the difficulties authors currently face specifying or communicating a multiverse analysis within their typical workflows play a role in discouraging more robust analysis. Our work, along with recent work in communicating multiverse analyses [20, 25], stand to contribute to the solution.

7 CONCLUSION

We contribute `multiverse`, a programming interface for multiverse analysis in R, which integrates with the RStudio IDE and RMarkdown documents to support a computational notebook workflow. Whereas previous tools for multiverse analysis [22, 37] only support script-style programming, `multiverse` enables a more iterative and interactive authoring experience consistent with the practices of data science workers [31, 32]. `multiverse` fills a gap in the rapidly evolving design space for multiverse programming interfaces by providing a *consistent fundamental operation*, `branch`, that is flexible enough to express any multiverse. We also contribute an evaluation of programming interfaces supporting multive rse analysis in R—`multiverse`, Boba [37], and mverse (an extension of `multiverse`)—where we identify trade-offs in design choices across the gamut of programming interfaces. We present opportunities and challenges for balancing trade-offs in API design and providing tools for debugging, authoring, and communicating multiverse analysis.

REFERENCES

[1] 2020 (accessed September 15, 2020). *Tidyverse*. https://www.tidyverse.org/

[2] Ruben C Arslan, Katharina M Schilling, Tanja M Gerlach, and Lars Penke. 2018. Using 26,000 diary entries to show ovulatory changes in sexual desire and behavior. *Journal of Personality and Social Psychology* (2018).

[3] Laura A Bakkensen and William Larson. 2014. Population matters when modeling hurricane fatalities. *Proceedings of the National Academy of Sciences* 111, 50 (2014), E5331–E5332.

[4] Henrik Bengtsson. 2020. *future: Unified Parallel and Distributed Processing in R for Everyone*. https://CRAN.R-project.org/package=future R package version 1.18.0.

[5] Donald Berry. 2012. Multiplicities in cancer research: ubiquitous and necessary evils. *Journal of the National Cancer Institute* 104, 15 (2012), 1125–1133.

[6] Alan F Blackwell, Carol Britton, A Cox, Thomas RG Green, Corin Gurr, Gada Kadoda, MS Kutar, Martin Loomes, Chrystopher L Nehaniv, Marian Petre, et al. 2001. Cognitive dimensions of notations: Design tools for cognitive technology. In *International Conference on Cognitive Technology*. Springer, 325–341.

[7] Christopher J Bryan, David S Yeager, and Joseph M O'Brien. 2019. Replicator degrees of freedom allow publication of misleading failures to replicate. *Proceedings of the National Academy of Sciences* 116, 51 (2019), 25535–25545.

[8] Joseph Cesario, David J Johnson, and William Terrill. 2019. Is there evidence of racial disparity in police use of deadly force? Analyses of officer-involved fatal shootings in 2015–2016. *Social psychological and personality science* 10, 5 (2019), 586–595.

[9] Björn Christensen and Sören Christensen. 2014. Are female hurricanes really deadlier than male hurricanes? *Proceedings of the National Academy of Sciences* 111, 34 (2014), E3497–E3498.

[10] Pasquale Cirillo and Nassim Nicholas Taleb. 2016. On the statistical properties and tail risk of violent conflicts. *Physica A: Statistical Mechanics and its Applications* 452 (2016), 29–45.

[11] Andy Cockburn, Carl Gutwin, and Alan Dix. 2018. Hark no more: on the preregistration of chi experiments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[12] J Anthony Cookson. 2018. When saving is gambling. *Journal of Financial Economics* 129, 1 (2018), 24–45.

[13] Marcus Credé and Leigh A Phillips. 2017. Revisiting the power pose effect: How robust are the results reported by Carney, Cuddy, and Yap (2010) to data analytic decisions? *Social Psychological and Personality Science* 8, 5 (2017), 493–499.

[14] Egon Dejonckheere, Elise K Kalokerinos, Brock Bastian, and Peter Kuppens. 2019. Poor emotion regulation ability mediates the link between depressive symptoms and affective bipolarity. *Cognition and Emotion* 33, 5 (2019), 1076–1083.

[15] Egon Dejonckheere, Merijn Mestdagh, Marlies Houben, Yasemin Erbas, Madeline Pe, Peter Koval, Annette Brose, Brock Bastian, and Peter Kuppens. 2018. The

bipolarity of affect and depressive symptoms. *Journal of personality and social psychology* 114, 2 (2018), 323.

[16] Marco Del Giudice and Steven Gangestad. 2020. A Traveler's Guide to the Multiverse: Promises, Pitfalls, and a Framework for the Evaluation of Analytic Decisions. *Advances in Methods and Practices in Psychological Science* (07 2020).

[17] Emilie Delaherche, Mohamed Chetouani, Ammar Mahdhaoui, Catherine Saint-Georges, Sylvie Viaux, and David Cohen. 2012. Interpersonal Synchrony: A Survey of Evaluation Methods across Disciplines. *IEEE Transactions on Affective Computing* 3, 3 (Jul 2012), 349–365. https://doi.org/10.1109/T-AFFC.2012.12

[18] Seamus Donnelly, Patricia J Brooks, and Bruce D Homer. 2019. Is there a bilingual advantage on interference-control tasks? A multiverse meta-analysis of global reaction time and interference cost. *Psychonomic bulletin & review* 26, 4 (2019), 1122–1147.

[19] Pierre Dragicevic. 2016. Fair statistical communication in HCI. In *Modern statistical methods for HCI*. Springer, 291–330.

[20] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the transparency of research papers with explorable multiverse analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–15.

[21] Jonah Gabry, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. 2019. Visualization in Bayesian workflow. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 182, 2 (2019), 389–402. https://doi.org/10.1111/rssa.12378

[22] Joachim Gassen. 2019. Researcher Degrees of Freedom Analysis (0.0.0.9). https://github.com/joachim-gassen/rdfanalysis

[23] Andrew Gelman and Eric Loken. 2013. The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time. *Department of Statistics, Columbia University* (2013).

[24] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[25] Brian D. Hall, Yang Liu, Yvonne Jansen, Pierre Dragicevic, Fanny Chevalier, and Matthew Kay. 2022. A Survey of Tasks and Visualizations in Multiverse Analysis Reports. *Computer Graphics Forum* 41, 1 (2022), 402–426. https://doi.org/10.1111/cgf.14443

[26] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 91–100.

[27] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[28] E. L. Hutchins, J. D. Hollan, and D. A. Norman. 1985. Direct manipulation interfaces. *Human-computer interaction* 1, 4 (1985), 311–338. https://doi.org/10.1207/s15327051hci0104_2

[29] Kiju Jung, Sharon Shavitt, Madhu Viswanathan, and Joseph M Hilbe. 2014. Female hurricanes are deadlier than male hurricanes. *Proceedings of the National Academy of Sciences* 111, 24 (2014), 8782–8787.

[30] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2917–2926.

[31] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.

[32] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 25–29.

[33] Mary Beth Kery and Brad A Myers. 2018. Interactions for untangling messy history in a computational notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 147–155.

[34] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–11.

[35] Donald E. Knuth. 1984. Literate Programming. *Comput. J.* 27, 2 (May 1984), 97–111. https://doi.org/10.1093/comjnl/27.2.97

[36] Sean Kross and Philip J Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.

[37] Yang Liu, Alex Kale, Tim Althoff, and Jeffrey Heer. 2020. Boba: Authoring and Visualizing Multiverse Analyses. *arXiv preprint arXiv:2007.05551* (2020).

[38] Steve Maley. 2014. Statistics show no evidence of gender bias in the public's hurricane preparedness. *Proceedings of the National Academy of Sciences* 111, 37 (2014), E3834–E3834.

[39] Daniel Malter. 2014. Female hurricanes are not deadlier than male hurricanes. *Proceedings of the National Academy of Sciences* 111, 34 (2014), E3496–E3496.

[40] James G March. 1991. Exploration and exploitation in organizational learning. *Organization science* 2, 1 (1991), 71–87.

[41] Philipp K. Masur and Michael Scharkow. 2019. specr: Statistical functions for conducting specification curve analyses (Version 0.2.1.9000). https://github.com/masurp/specr

[42] Michael Jongho Moon, Haoda Li, Mingwei Xu, Nathan Taback, Fanny Chevalier, and Alison Gibbs. 2022. *mverse: Tidy Multiverse Analysis Made Simple*. https://cran.r-project.org/web/packages/mverse/index.html R package version 0.1.0.

[43] Deborah Nolan and Duncan Temple Lang. 2007. Dynamic, interactive documents for teaching statistical practice. *International Statistical Review* 75, 3 (2007), 295–321.

[44] Donald A. Norman. 2002. *The Design of Everyday Things*. Basic Books, Inc., USA.

[45] Brian A Nosek, Charles R Ebersole, Alexander C DeHaven, and David T Mellor. 2018. The preregistration revolution. *Proceedings of the National Academy of Sciences* 115, 11 (2018), 2600–2606.

[46] Amy Orben and Andrew K Przybylski. 2019. The association between adolescent well-being and digital technology use. *Nature Human Behaviour* 3, 2 (2019), 173–182.

[47] Chirag J Patel, Belinda Burford, and John PA Ioannidis. 2015. Assessment of vibration of effects due to model specification can demonstrate the instability of observational associations. *Journal of clinical epidemiology* 68, 9 (2015), 1046–1058.

[48] Gregory J Poarch, Jan Vanhove, and Raphael Berthele. 2019. The effect of bidialectalism on executive function. *International Journal of Bilingualism* 23, 2 (2019), 612–628.

[49] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[50] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[51] Daniel J. Schad, Michael Betancourt, and Shravan Vasishth. 2021. Toward a principled Bayesian workflow in cognitive science. *Psychological Methods* 26, 1 (2021), 103–126. https://doi.org/10.1037/met0000275

[52] Beau Sheil. 1986. Power tools for programmers. In *Readings in artificial intelligence and software engineering*. 573–580.

[53] Raphael Silberzahn, Eric Luis Uhlmann, Daniel P Martin, Pasquale Anselmi, Frederik Aust, Eli Awtrey, Štěpán Bahník, Feng Bai, Colin Bannard, Evelina Bonnier, et al. 2018. Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science* 1, 3 (2018), 337–356.

[54] Joseph P Simmons, Leif D Nelson, and Uri Simonsohn. 2011. False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological science* 22, 11 (2011), 1359–1366.

[55] Uri Simonsohn, Joseph P. Simmons, and Leif D. Nelson. 2020. Specification curve analysis. *Nature Human Behaviour* 4, 11 (Nov 2020), 1208–1214. https://doi.org/10.1038/s41562-020-0912-z

[56] Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. 2016. Increasing transparency through a multiverse analysis. *Perspectives on Psychological Science* 11, 5 (2016), 702–712.

[57] Martin Voracek, Michael Kossmeier, and Ulrich S Tran. 2019. Which Data to Meta-Analyze, and How? *Zeitschrift für Psychologie* (2019).

[58] Hadley Wickham. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

[59] Hadley. Wickham. 2019. *Advanced R* (2nd edition ed.). CRC Press, Boca Raton, FL, Chapter 7.

[60] Hadley. Wickham. 2019. *Advanced R* (2nd edition ed.). CRC Press, Boca Raton, FL, Chapter 19.

[61] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2018. Design Exposition with Literate Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2018), 759–768.

[62] Cristobal Young and Katherine Holsteen. 2017. Model uncertainty and robustness: A computational framework for multimodel analysis. *Sociological Methods & Research* 46, 1 (2017), 3–40.