Tea: A High-level Language and Runtime System for Automating Statistical Analysis

Eunice Jun¹, Maureen Daum¹, Jared Roesch¹, Sarah Chasins², Emery Berger³,⁴, Rene Just¹, Katharina Reinecke¹

¹University of Washington, Seattle, WA
{emjun, mdaum, jroesch, rjust, reinecke}@uw.edu

²University of California, Berkeley, CA
schasins@cs.berkeley.edu

³University of Massachusetts Amherst,
emery@cs.umass.edu

ABSTRACT

Though statistical analyses are centered on research questions and hypotheses, current statistical analysis tools are not. Users must first translate their hypotheses into specific statistical tests and then perform API calls with functions and parameters. To do so accurately requires that users have statistical expertise. To lower this barrier to valid, replicable statistical analysis, we introduce Tea, a high-level declarative language and runtime system. In Tea, users express their study design, any parametric assumptions, and their hypotheses. Tea compiles these high-level specifications into a constraint satisfaction problem that determines the set of valid statistical tests and then executes them to test the hypothesis. We evaluate Tea using a suite of statistical analyses drawn from popular tutorials. We show that Tea generally matches the choices of experts while automatically switching to non-parametric tests when parametric assumptions are not met. We simulate the effect of mistakes made by non-expert users and show that Tea automatically avoids both false negatives and false positives that could be produced by the application of incorrect statistical tests.

Author Keywords
statistical analysis; automated statistical analysis; declarative programming language; constraint-based system; data science; reproducibility; pre-registration

CCS Concepts
•Human-centered computing → User interface toolkits;

INTRODUCTION

The enormous variety of modern quantitative methods leaves researchers with the nontrivial task of matching analysis and design to the research question.

- Ronald Fisher [16]

Since the development of modern statistical methods (e.g., Student’s t-test, ANOVA, etc.), statisticians have acknowledged the difficulty of identifying which statistical tests people should use to answer their specific research questions. Almost a century later, choosing appropriate statistical tests for evaluating a hypothesis remains a challenge. As a consequence, errors in statistical analyses are common [26], especially given that data analysis has become a common task for people with little to no statistical expertise.

A wide variety of tools (such as SPSS [55], SAS [54], and JMP [52]), programming languages (e.g., R [53]), and libraries (including numpy [40], scipy [23], and statsmodels [45]), enable people to perform specific statistical tests, but they do not address the fundamental problem that users may not know which statistical test to perform and how to verify that specific assumptions about their data hold.

In fact, all of these tools place the burden of valid, replicable statistical analyses on the user and demand deep knowledge of statistics. Users not only have to identify their research questions, hypotheses, and domain assumptions, but also must select statistical tests for their hypotheses (e.g., Student’s t-test or one-way ANOVA). For each statistical test, users must be aware of the statistical assumptions each test makes about the data (e.g., normality or equal variance between groups) and how to check for them, which requires additional statistical tests (e.g., Levene’s test for equal variance), which themselves may demand further assumptions about the data. This cognitively demanding process requires significant knowledge about statistical tests and their preconditions as well as the ability to perform the tests and verify their preconditions. This process can easily lead to mistakes.

This paper presents Tea¹, a high-level declarative language for automating statistical test selection and execution that abstracts the details of statistical analysis from the users. Tea captures users’ hypotheses and domain knowledge, translates this information into a constraint satisfaction problem, identifies all valid statistical tests to evaluate a hypothesis, and executes the tests. Tea’s higher-level, declarative nature aims to lower the barrier to valid, replicable analyses.

¹named after Fisher’s “Lady Tasting Tea” experiment [16]
We have designed Tea to integrate directly into common data analysis workflows for users who have minimal programming experience. Tea is implemented as an open-source Python library, so programmers can use Tea wherever they use Python, including within Python notebooks.

In addition, Tea is flexible. Its abstraction of the analysis process and use of a constraint solver to select tests is designed to support its extension to emerging statistical methods, such as Bayesian analysis. Currently, Tea supports frequentist Null Hypothesis Significance Testing (NHST).

The paper makes the following contributions:

• Tea, a novel domain-specific language (DSL) for automatically selecting and executing statistical analyses based on users’ hypotheses and domain knowledge (Section 5),

• the Tea runtime system, which formulates statistical test selection as a maximum constraint satisfaction problem (Section 6), and

• an initial evaluation showing that Tea can express and execute common NHST statistical tests (Section 7).

We start with a usage scenario that provides an overview of Tea (Section 2). We discuss the concerns about statistics in the HCI community that shaped Tea’s design (Section 3), the implementation of Tea’s programming language (Section 5), the implementation of Tea’s runtime system (Section 6), and the evaluation of Tea as a whole (Section 7). We discuss limitations and future work, differentiate Tea from prior work, and conclude with information on how to use Tea.

USAGE SCENARIO
This section describes how an analyst who has no statistical background can use Tea to answer their research questions. We use as an example analyst a historical criminologist who wants to determine how imprisonment differed across regions of the US in 1960. Figure 1 shows the Tea code for this example.

The analyst specifies the data file’s path in Tea. Tea handles loading and storing the data set for the duration of the analysis session. The analyst does not have to worry about reformatting the data during the analysis process in any way.

The analyst asks if the probability of imprisonment was higher in southern states than in non-southern states. The analyst identifies two variables that could help them answer this question: the probability of imprisonment (‘Prob’) and geographic location (‘So’). Using Tea, the analyst defines the geographic location as a dichotomous nominal variable where ‘1’ indicates a southern state and ‘0’ indicates a non-southern state, and indicates that the probability of imprisonment is a numeric data type (ratio) with a range between 0 and 1.

The analyst then specifies their study design, defining the study type to be “observational study” (rather than “experimental study”) and defining the contributor (independent) variable to be the geographic location and the outcome (dependent) variable to be the probability of imprisonment.

Based on their prior research, the analyst knows that the probability of imprisonment in southern and non-southern states is normally distributed. The analyst provides an assumptions clause to Tea in which they specify this domain knowledge. They also specify an acceptable Type I error rate (probability of finding a false positive result), more colloquially known as the “significance threshold” (\( \alpha = .05 \)) that is acceptable in criminology. If the analyst does not have assumptions or forgets to provide assumptions, Tea will use the default of \( \alpha = .05 \).

The analyst hypothesizes that southern states will have a higher probability of imprisonment than non-southern states. The analyst directly expresses this hypothesis in Tea. Note that at no point does the analyst indicate which statistical tests should be performed.

From this point on, Tea operates entirely automatically. When the analyst runs their Tea program, Tea checks properties of the data and finds that the Student’s t-test is appropriate. Tea executes the Student’s t-test and non-parametric alternatives, such as the Mann-Whitney U test, which provide alternative, consistent results.

Tea generates a table of results from executing the tests, ordered by their power (i.e., results from the parametric t-test will be listed first given that it has higher power than the non-parametric equivalent). Based on this output, the analyst concludes that their hypothesis—that the probability of im-

---

2 The example is taken from Ehrlich [13] and Vandaele [47]. The data set comes as part of the MASS package in R.
prisonment was higher in southern states than in non-southern states in 1960—is supported. The results from alternative statistical tests support this conclusion, so the analyst can be confident in their assessment.

The analyst can now share their Tea program with colleagues. Other researchers can easily see what assumptions the analyst made and what the intended hypothesis was (since these are explicitly stated in the Tea program), and reproduce the exact results using Tea.

**DESIGN CONSIDERATIONS**

In designing Tea’s language and runtime system, we considered best practices for conducting statistical analyses and derived our own insights on improving the interaction between users and statistical tools.

We identified five key recommendations for statistical analysis from Cairns’ report on common statistical errors in HCI [6], which echoes many concerns articulated by Wilkinson [56], and the American Psychological Association’s Task Force on Statistical Inference [1]:

1. **Users should be able to express their expertise, assumptions, and intentions for analysis.** Users have domain knowledge and goals that cannot be expressed with the low-level API calls to the specific statistical tests required by the majority of current tools. A higher level of abstraction that focuses on the goals and context of analysis is likely to appeal to users who may not have statistical expertise (Section 5).

2. **Users should not be burdened with statistical details to conduct valid analyses.** Currently, users must not only remember their hypotheses but also identify possibly appropriate tests and manually check the preconditions for all the tests. Simplifying the user’s procedure by automating the test selection process can help reduce cognitive demand (Section 6).

3. **Inspect data properties and infer valid statistical tests.** While there are calls to incorporate other methods of statistical analysis [27, 26], Null Hypothesis Significance Testing (NHST) remains the norm in HCI and other disciplines. Therefore, Tea currently implements a module for NHST with the tests found to be most common by [50] (see subsection 6.6 for a list of tests). We believe that Tea’s abstraction and modularity will enable the incorporation of other statistical analysis approaches as they move into the mainstream.

**OVERVIEW OF TEA**

Tea consists of a high-level programming language and a runtime system. There are three key steps to compiling a Tea program from user specifications to executing statistical analyses:

1. **Check for completeness and syntax.** Tea first checks that a user’s program specifies a data set, variable declarations, study design description, a set of assumptions, and hypotheses using the correct syntax. For pre-registration (further discussed in Section 9), the data set can be empty (with only column names). If there are any syntax errors or missing parts, Tea will issue an error and stop execution.

2. **Check for consistent, well-formed hypotheses.** Using the variable declarations, Tea then checks that the hypotheses the user states are consistent with variable data types. For instance, Tea would issue an error and halt execution if a nominal variable was hypothesized to have a positive relationship with another nominal variable. If the nominal variables have categories given by numbers (e.g., a variable for education where ‘1’ stands for ‘High School’, ‘2’ for ‘College’, etc.), a linear relationship would be possible to compute by treating the categories as raw continuous values. However, treating the numbers as values is incorrect and the results misleading because the numbers represent discrete categories, not continuous values. Tea avoids such mistakes.

3. **Inspect data properties and infer valid statistical tests.** Once Tea’s compiler verifies that a Tea program is complete, syntactically correct, and consistent, Tea’s runtime system inspects the data to verify properties about it and find a set of valid statistical tests. The higher-level Tea program is then compiled to logical constraints, which is further discussed in Section 6.

**TEA’S PROGRAMMING LANGUAGE**

Tea is a domain-specific language embedded in Python. It takes advantage of existing Python data structures (e.g., classes, dictionaries, and enums). We chose Python because of its widespread adoption in data science. Tea is itself implemented as a Python library.

A key challenge in describing studies is determining the level of granularity necessary to produce an accurate analysis. In Tea programs, users describe their studies in five ways: (1) providing a data set, (2) describing the variables of interest in that data set, (3) specifying their study design, (4) stating their assumptions about the variables, and (5) formulating hypotheses about the relationships between variables.

---

3 Tea is open-source and available for download on pip, a common Python package manager.
Table 1: Comparison of Tea to other tools. Despite the published best practices for statistical analyses, most tools do not help users select appropriate tests. Tea not only addresses the best practices but also supports reproducing analyses.

<table>
<thead>
<tr>
<th>Best practices</th>
<th>SAS</th>
<th>SPSS</th>
<th>JMP</th>
<th>R</th>
<th>Statsplorer [50]</th>
<th>Tea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit statement of user assumptions</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Automatic verification of test preconditions</td>
<td>—</td>
<td>—</td>
<td>sometimes</td>
<td>sometimes</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Automatic accounting of multiple comparisons</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>sometimes</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Surface alternative analyses</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Contextualize results</td>
<td>✓</td>
<td>sometimes</td>
<td>✓</td>
<td>sometimes</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Easy to reproduce analysis</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
</tbody>
</table>

Test: students_t
***Test assumptions:
Exactly two variables involved in analysis: So, Prob
Exactly one explanatory variable: So
Exactly one explained variable: Prob
Independent (not paired) observations: So
Variable is categorical: So
Variable has two categories: So
Continuous (not categorical) data: Prob
Equal variance: So, Prob
Groups are normally distributed: So, Prob

***Test results:
name = Student's T Test
test_statistic = 4.20213
p_value = 0.00012
adjusted_p_value = 0.00006
alpha = 0.05
dof = 45
Effect size:
Cohen's d = 1.24262
A12 = 0.83669
Null hypothesis = There is no difference in means between So = 0 and So = 1 on Prob
Interpretation = (45) = 4.20213, p = 0.00006. Reject the null hypothesis at alpha = 0.05. The mean of Prob for So = 1 (M=0.06371, SD=0.02251) is significantly greater than the mean for So = 0 (M=0.03851, SD=0.01778). The effect size is Cohen's d = 1.24262, A12 = 0.83669. The effect size is the magnitude of the difference, which gives a holistic view of the results [1].

Figure 2: Part of Tea’s output. The output is a result of running the sample program in Figure 1. Tea outputs the data properties that led Tea to select the statistical test as well as results from executing the test, effect size calculations, the null hypothesis tested, and the interpretation of the results, which can be included in publications with minor editing.

Data
Data is required for executing statistical analyses. One challenge in managing data for analysis is minimizing both duplicated data and user intervention.

To reduce the need for user intervention for data manipulation, Tea requires the data to be a CSV in long format. CSVs are a common output format for data storage and cleaning tools. Long format (sometimes called “tidy data” [51]) is a denormalized format that is widely used for collecting and storing data, especially for within-subjects studies.

Unlike R and Python libraries such as numpy [40], Tea only requires one instance of the data. Users do not have to duplicate the data or subsets of it for analyses that require the data to be in slightly different forms. Minimizing data duplication or segmentation is also important to avoid user confusion about where some data exist or which subsets of data pertain to specific statistical tests.

Optionally, users can also indicate a column in the data set that acts as a relational (or primary) key, or an attribute that uniquely identifies rows of data. For example, this key could be a participant identification number in a behavioral experiment. A key is useful for verifying a study design, described below. Without a key, Tea’s default is that all rows in the data set comprise independent observations (that is, all variables are between subjects).

For pre-registration where there is no data, a CSV with only column names is necessary. Using Tea for pre-registration is discussed further in Section 9.

Variables
Variables represent columns of interest in the data set. Variables have a name, a data type (nominal, ordinal, interval, or ratio), and, when appropriate, valid categories. Users (naturally) refer to variables through a Tea program using their names. Only nominal and ordinal variables have a list of possible categories. For ordinal variables, the categories are also ordered from left to right.

Variables encapsulate queries. The queries represent the index of the variable’s column in the original data set and any filtering operations applied to the variable. For instance, it is common to filter by category for nominal variables.

Study Design
Three aspects of study design are important for conducting statistical analyses: (1) the type of study (observational study vs. randomized experiment), (2) the independent and dependent variables, and (3) the number of observations per participant (e.g., between-subjects variables vs. within-subjects variables).

For semantic precision, Tea uses different terms for independent and dependent variables for observational studies and experiments. In experiments, variables are described as either “independent” or “dependent” variables. In observational studies, variables are either “contributor” (independent) or “outcome” (dependent) variables.

Assumptions
Users’ assumptions based on domain knowledge are critical for conducting and contextualizing studies and analyses. Often, users’ assumptions are particular to variables and specific properties (e.g., equal variances across different groups). Current tools generally do not require that users encode these assumptions, leaving them implicit.

Tea takes the opposite approach to contextualize and increase the transparency of analyses. It requires that users be explicit about assumptions and statistical properties pertaining to the
import tea
tes.data('stataex77.csv')
variables = {
    'name': 'Illiteracy',
    'data type': 'interval',
    'categories': [0, 100]
}
variables = {
    'name': 'HS Grad',
    'data type': 'ratio',
}
variables = {
    'name': 'Life Exp',
    'data type': 'ratio',
}
}
tea.define_variables(variables)

study_design = {
    'study type': 'observational study',
    'contributor variables': ['Illiteracy', 'HS Grad'],
    'outcome variables': 'Life Exp'
}
tea.define_study_design(study_design)

assumptions = {
    'Type I (False Positive) Error Rate': 0.05,
    'normal distribution': ['Illiteracy']
}
tea.assume(assumptions)
tea.hypothesize(['Illiteracy', 'Life Exp'], ['Illiteracy - Life Exp'])

Figure 3: **Tea program and its mode-dependent executions.**

a) Tea program that aims to determine if two contributor variables, 'Illiteracy' and 'HS Grad' that may predict a third outcome variable 'Life Exp', are correlated. The user asserts that 'Illiteracy' is normally distributed.  
b) By default, Tea executes programs in the **strict** mode.  
c) Warning that Tea disagrees with the user and will override the user's assertion that 'Illiteracy' is normally distributed in the **strict** mode.  
d) Results without the parametric test since Tea overrides user's assertion.  
e) A single line change can modify Tea to execute a program in the **relaxed** mode.  
f) Warning that Tea cannot verify normality for 'Illiteracy' but will defer to user's assertion.  
g) Results with the parametric test since Tea proceeds as if 'Illiteracy' was normally distributed.

analysis as a whole (e.g., acceptable Type I error rate/significance threshold) and the data.

Tea supports two modes for treating user assumptions: **strict** and **relaxed**. In both modes, Tea verifies all user assumptions and issues warnings for assumptions that statistical testing does not verify. In the **strict** mode, Tea overrides user assumptions when selecting valid statistical tests. In the **relaxed** mode, Tea defers to user assumptions and proceeds as if the assumptions verified even if they did not. The **strict** mode is the default, but users can specify the **relaxed** mode. Figure 3 shows the two modes and the different warnings and output they generate.

If users also know that a data transformation (i.e., log transformation) applies to a variable, they can express this as an assumption. Data transformations are not properties to be verified but rather treatments of data that are applied during assumption verification, statistical test selection, and test execution, which is why they are included in the assumptions clause. The next section discusses the verification process for assumptions in greater detail.

**Hypotheses**

Hypotheses drive the statistical analysis process. Users often have hypotheses that are technically alternative hypotheses. Tea focuses on capturing users’ alternative hypotheses about the relationship between two or more variables. Tea uses the alternate hypothesis to conduct either a two-sided or one-sided statistical test. By default, Tea uses the null hypothesis that there is no relationship between variables.

Figure 4 exemplifies the range of hypotheses Tea supports.

**TEA’S RUNTIME SYSTEM**

Tea compiles programs into logical constraints about the data and variables, which it resolves using a constraint solver. A significant benefit of using a constraint solver is extensibility. Adding new statistical tests does not require modifying the core of Tea’s runtime system. Instead, defining a new test
We derived preconditions for tests from an online HCI and its constraint solver, Tea uses Z3 [10], a Satisfiability Modulo Theory (SMT) solver.

At runtime, Tea invokes a solver that operates on the logical constraints it computes to produce a list of valid statistical tests to conduct. This process presents three key technical challenges: (1) incorporating statistical knowledge as constraints, (2) expressing user assumptions as constraints, and (3) recursively selecting statistical tests to verify preconditions of other statistical tests.

**SMT Solver**

As its constraint solver, Tea uses Z3 [10], a Satisfiability Modulo Theory (SMT) solver.

Satisfiability is the process of finding an assignment to variables that makes a logical formula true. For example, given the logical rules \(0 < x < 100\) and \(y < x\), \(\{x = 1, y = 0\}\), \(\{x = 10, y = 5\}\), and \(\{x = 99, y = -100\}\) would all be valid assignments that satisfy the rules. SMT solvers determine the satisfiability of logical formulas, which can encode boolean, integer, real number, and uninterpreted function constraints over variables. SMT solvers can also be used to encode constraint systems, as we use them here. A wide variety of applications ranging from the synthesis of novel interface designs [46], the verification of website accessibility [41], and the synthesis of data structures [33] employ SMT solvers.

**Logical Encodings**

The first challenge of framing statistical test selection as a constraint satisfaction problem is defining a logical formulation of statistical knowledge.

Tea encodes the applicability of a statistical test based on its preconditions. A statistical test is applicable if and only if all of its preconditions (which are properties about variables) hold. We derived preconditions for tests from an online HCI and statistics course [29], a statistics textbook [15], and publicly available data science resources from universities [4, 32].

Tea represents each precondition for a statistical test as an uninterpreted function representing a property over one or more variables. Each property is assigned `true` if the property holds for the variable/s; similarly, if the property does not hold, the property function is assigned `false`.

Tea also encodes statistical knowledge about variable types and properties that are essential to statistical analysis as axioms, such as the constraint that only a continuous variable can be normally distributed.

**Algorithm**

Tea frames the problem of finding a set of valid statistical tests as a maximum satisfiability (MaxSAT) problem that is seeded with user assumptions.

First, Tea translates each user assumption about a data property into an axiom about a property and variable. As described in subsection 5.4, user assumptions about properties but not data transformations are always checked. In the strict mode, Tea overrides any user assumptions it does not find to hold, creating an axiom that a property is `false`. In the relaxed mode, Tea defers to user assumptions, creating axioms that a property is `true`. For any user assumptions that do not pass statistical testing, Tea warns the user and explains how it will proceed depending on the mode.

Then, for each new statistical test Tea tries to satisfy, Tea checks to see if each precondition holds. For each precondition checked, Tea adds the property and variable checked as an axiom to observe as future tests are checked. If any property violates the axioms derived from users’ assumptions, the property is removed and the test is invalidated. Users’ assumptions always take precedence.

The constraint solver then prunes the search space. Tea does not compute all properties for all variables, a significant optimization when analyzing very large data sets.

At the end of this process, Tea finds a set of valid statistical tests to execute. If this set is empty, Tea defaults to its implementation of bootstrapping [12]. Otherwise, Tea proceeds and executes all valid statistical tests. Tea returns a table of results to users, applying multiple comparison corrections [22] and calculating effect sizes when appropriate.

**Optimization: Recursive Queries**

When Tea verifies a property holds for a variable, it often must invoke another statistical test. For example, to check that two groups have equal variance, Tea must execute Levene’s test. The statistical test used for verification may then itself have a precondition, such as a minimum sample size.

Such recursive queries are inefficient for SMT solvers like Z3 to reason about. To eliminate recursion, Tea lifts some statistical tests to properties. For instance, Tea does not encode the Levene’s test as a statistical test. Instead, Tea encodes statistical tests to properties. For instance, Tea does not encode statistical tests to properties. For instance, Tea does not encode the Levene’s test as a statistical test. Instead, Tea encodes the property of having equal variance between groups and executes the Levene’s test for two groups when verifying that property for particular variables.

**User Output**

The result of running a Tea program with data is a listing of the results of executing valid statistical tests, as shown in Figure 2. For each valid statistical test executed, the output contains the properties of data that Tea checked and used to determine that a statistical test applied, the test statistic value, p-value (and an adjusted p-value, if applicable), effect sizes (Cohen’s \(d\) [9] and Vargha Delaney A12 [48]), the alpha level the user specified in their program, the precise null hypothesis the statistical test
We differentiate between candidate and valid tests. A candidate test can be computed on the data, when ignoring any preconditions regarding the data types or distributions. A valid test is a candidate test for which all preconditions are satisfied.

**Null Hypothesis Significance Testing Module**

Tea currently implements tests common to NHST in HCI. In particular, Tea supports four classes of tests: correlation (parametric: Pearson’s $r$, Pointbiserial; non-parametric: Kendall’s $τ$, Spearman’s $ρ$), bivariate mean comparison (parametric: Student’s $t$-test, Paired $t$-test; non-parametric: Mann-Whitney U, Wilcoxon signed rank, Welch’s), multivariate mean comparison (parametric: F-test, Repeated measures one way ANOVA, Factorial ANOVA, Two-way ANOVA; non-parametric: Kruskal Wallis, Friedman), and comparison of proportions (Chi Square, Fisher’s Exact). Tea also supports an implementation of bootstrapping [12].

**INITIAL EVALUATION**

We assessed the validity of Tea in two ways. First, we compared Tea’s suggestions of statistical tests to suggestions in textbook tutorials. We use these tutorials as a proxy for expert test selection. Second, for each tutorial, we compared the analysis results of the test(s) suggested by Tea to those of the test suggested in the textbook as well as all other candidate tests. We use the set of all candidate tests as a proxy for non-expert test selection.

We use these approaches to evaluate whether Tea avoids common mistakes made by non-expert users and to understand the trade-offs between statistical tests and visualizations.

**How does Tea compare to textbook tutorials?**

Our goal was to compare Tea to expert recommendations. We sampled 12 data sets and examples from R tutorials ([24] and [15]). These included eight parametric tests, four non-parametric tests, and one Chi-square test. We chose these tutorials because they appeared in two of the top 20 statistical textbooks on Amazon and had publicly available data sets, which did not require extensive data wrangling.

We translated all analyses into Tea and encoded any assumptions explicitly stated in the tutorial. Tea selected tests based only on the data and the assumptions expressed in the Tea program. Where Tea disagreed with the tutorials, either (1) the tutorial authors chose the wrong analyses or (2) the tutorial authors had implicit assumptions about the data that did not hold up to statistical testing.

For nine out of the 12 tutorials, Tea suggested the same statistical test (see Table 2). For three out of 12 tutorials, which used a parametric test, Tea suggested using a non-parametric alternative instead. Tea’s recommendation of using a non-parametric test instead of a parametric one did not change the statistical significance of the result at the .05 level. Tea suggested non-parametric tests based on the Shapiro-Wilk test for normality. It is possible that tutorial authors visualized the data to make implicit assumptions about the data, but this practice or conclusion was not made explicit in the tutorials. (We discuss the trade-offs between statistical tests and visualizations for testing data properties in Section 8.)

For the two-way ANOVA tutorial from [15], which studied how gender and drug usage of individuals affected their perception of attractiveness, a precondition of the two-way ANOVA is that the dependent measure is normally distributed in each category. This precondition was violated. As a result, Tea defaulted to bootstrapping the means for each group and reported the means and confidence intervals. For the pointbiserial correlation tutorial from [15], Tea also defaulted to bootstrap for two reasons. First, the precondition of normality is violated. Second, the data uses a dichotomous (nominal) variable, which invalidates Spearman’s $ρ$ and Kendall’s $τ$.

Tea generally agrees with expert recommendations and is more conservative in the presence of non-normal data, minimizing the risk of false positive findings.

**Does Tea avoid common mistakes made by non-expert users?**

Our goal was to assess whether any of the tests suggested by Tea (i.e., valid candidate tests) or any of the invalid candidate tests would lead to a different conclusion than the one drawn in the tutorial. Table 2 shows the results. Specifically, emphasized $p$-values indicate instances for which the result of a test differs from the tutorial in terms of statistical significance at the .05 level.

For all of the 12 tutorials, Tea’s suggested tests led to the same conclusion about statistical significance. For two out of the 12 tutorials, two or more candidate tests led to a different conclusion. These candidate tests were invalid due to violations of independence or normality.

To summarize, the evaluation shows us that (i) Tea can replicate and even improve upon expert choices and (ii) Tea could help novices avoid common mistakes and false conclusions.

**LIMITATIONS AND FUTURE WORK**

The goal of this paper was to design and assess Tea’s high-level DSL and constraint-based runtime system. Here, we identify limitations of the current work that suggest opportunities for future work.

**Empirical evaluation of usability.** While we believe that abstracting away statistical tests—thus obviating the need for detailed statistical knowledge—will make Tea easier to use than conventional statistical tools, an empirical evaluation with non-statistical expert users is required to establish this. A study comparing Tea to conventional statistical analysis tools such as SPSS or R would be of particular interest.

Dichotomous thinking, such as relying on the result of a significance test to definitively decide if there is evidence to support a hypothesis, ignores the need to consider the magnitude of effects. Practical significance is more important than statistical significance for decision making, so tools should support reasoning about practical significance. Tea aims to guide users away from dichotomous thinking and towards holistic interpretations of analyses by providing effect sizes, clear statements...
Table 2: Results of applying Tea to 12 textbook tutorials.

Tea is comparable to an expert selecting statistical tests. Tea can prevent false positive and false negative results by suggesting only tests that satisfy all assumptions. *Tutorial* gives the test described in the textbook; *Candidate tests (p-value)* gives all tests a user could run on the provided data with corresponding p-values; *Assumptions* gives all satisfied (lightly shaded) and violated (white) assumptions; *Tea suggests* indicates which tests Tea suggests based on their preconditions (assumptions about the data). Emphasized p-values indicate instances where a candidate test leads to a wrong conclusion about statistical significance. Although this table focuses on p-values, Tea produces richer output that provides a more holistic view of the statistical analysis results by including effect sizes, for instance. Refer to Figure 2 for an example of output from a Tea program.

<table>
<thead>
<tr>
<th>Tutorial</th>
<th>Candidate tests (p-value)</th>
<th>Assumptions*</th>
<th>Tea suggests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pearson</td>
<td>Pearson’s r (6.96925e-06)</td>
<td>3 3 5</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Kendall’s τ (2.04198e-05)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Spearman’s ρ (2.83575e-05)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td>Spearman’s ρ</td>
<td>Spearman’s ρ (.00172)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Pearson’s r (.01115)</td>
<td>3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Kendall’s τ (.00126)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td>Kendall’s τ</td>
<td>Kendall’s τ (.00126)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Pearson’s r (.01115)</td>
<td>3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Spearman’s ρ (.00172)</td>
<td>3 3</td>
<td>✓</td>
</tr>
<tr>
<td>Pointbiserial</td>
<td>Pointbiserial (Pearson’s r) (.00287)</td>
<td>3 3 5</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Spearman’s ρ (.00477)</td>
<td>3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Kendall’s τ (.00574)</td>
<td>3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Bootstrap (&lt;0.05)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Student’s t-test</td>
<td>Student’s t-test (.00012)</td>
<td>3 3 3 3 6 7 8</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Mann-Whitney U (9.27319e-05)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Welch’s t-test (.00065)</td>
<td>3 3 3 3 8</td>
<td>✓</td>
</tr>
<tr>
<td>Paired t-test</td>
<td>Paired t-test (.03098)</td>
<td>3 3 3 3 7 8</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Student’s t-test (.10684)</td>
<td>3 3 3 3 7 8</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Mann-Whitney U (.06861)</td>
<td>3 3 3 3 7</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Wilcoxon signed rank (.04586)</td>
<td>3 3 3 3 7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Welch’s t-test (.10724)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td>Wilcoxon signed rank</td>
<td>Wilcoxon signed rank (.04657)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Student’s t-test (.02690)</td>
<td>3 3 3 3 7</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Paired t-test (.01488)</td>
<td>3 3 3 3 7 8</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Mann-Whitney U (.00560)</td>
<td>3 3 3 3 7</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Welch’s t-test (.03572)</td>
<td>3 3 3 3</td>
<td>—</td>
</tr>
<tr>
<td>F-test</td>
<td>F-test (9.81852e-13)</td>
<td>3 3 3 3 6 9</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Kruskal Wallis (2.23813e-07)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Friedman (8.66714e-07)</td>
<td>3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Factorial ANOVA (9.81852e-13)</td>
<td>3 3 3 3 6 9</td>
<td>✓</td>
</tr>
<tr>
<td>Kruskal Wallis</td>
<td>Kruskal Wallis (.03419)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>F-test (.05578)</td>
<td>3 3 3 3 5 9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Friedman (3.02610e-08)</td>
<td>3 3 3 3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Factorial ANOVA (.05578)</td>
<td>3 3 3 3 5 9</td>
<td>✓</td>
</tr>
<tr>
<td>Repeated measures one way ANOVA</td>
<td>Repeated measures one way ANOVA (.0000)</td>
<td>3 3 3 3 5 9</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Kruskal Wallis (4.51825e-06)</td>
<td>3 3 3 3 5 9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>F-test (1.24278e-07)</td>
<td>3 3 3 3 6 9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Friedman (5.23589e-11)</td>
<td>3 3 3 3 9</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Factorial ANOVA (1.24278e-07)</td>
<td>3 3 3 3 6 9</td>
<td>✓</td>
</tr>
<tr>
<td>Two-way ANOVA</td>
<td>Two-way ANOVA (3.70282e-17)</td>
<td>3 3 3 3 5 9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Bootstrap (&lt;0.05)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Chi Square</td>
<td>Chi Square (4.76743e-07)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Fisher’s Exact (4.76743e-07)</td>
<td>3 3 3 3</td>
<td>✓</td>
</tr>
</tbody>
</table>

*① one variable, ② two variables, ③ two or more variables, ④ continuous vs. categorical vs. ordinal data, ⑤ normality, ⑥ equal variance, ⑦ dependent vs. independent observations, ⑧ exactly two groups, ⑨ two or more groups
about the null hypotheses tested in each statistical test, the results of the statistical tests in light of the specific null hypothesis, and citations for more statistical information (see Figure 2). However, Tea does not prevent partial reporting, or “cherry-picking” [11], of results. Further research is needed to investigate how automated systems show the results of multiple analyses, perhaps with interactive scaffolding of results to avoid cherry-picking.

Relaxing Tea’s conservatism. A conservative in its test selection because Tea’s runtime system will execute a statistical test only when all the preconditions are met. In practice, some preconditions may be more important than others. For instance, Tea could allow some degree of deviation from absolute normality. Further evaluation with statistical and domain experts could help refine Tea’s decision making procedure.

The benefit of implementing Tea’s runtime system with constraints is that its knowledge base can expand and become more refined as best practices for statistics evolve. Tea could employ a more sophisticated constraint system where, for instance, normality is treated as a soft constraint that has more weight when the sample size is small but is relaxed with a sufficiently large sample size. Still, normality tests may not be (a) ideal due to their over-sensitivity to small deviations from normality with large samples and their lack of power for small samples or (b) always necessary [34]. Therefore, a combination of statistical and graphical approaches to judging normality is likely best, as discussed next.

Moreover, because sample size and acceptable degrees of violating assumptions are highly domain dependent, Tea could also be expanded to incorporate reconfigurable domain-specific constraints. A promising direction is to imagine formalized domain-specific statistics guidelines. Similar to how \LaTeX uses style classes for typesetting text, Tea could take as input domain-specific guides for preferential weighting of properties and statistical tests.

Incorporating visualizations and humans-in-the-loop. To ascertain properties about the data (e.g., normality), Tea uses statistical tests rather than visualizations. There are trade-offs and limitations to both approaches. Statistical tests may be sensitive to sample size and slight violations, which is acceptable for many hypothesis tests and models [34]. The main advantage of statistically checking data properties is that decisions are reproducible and more objective. Through visual inspection, trained analysts can examine the data, detect and assess any violations to data properties that are unacceptable in their domain, and notice other properties about the data to check. As visualizations form an important part of analysts’ workflows for building models [17] and generating hypotheses [5], future work should investigate incorporating visualizations with Tea for a human-in-the-loop system that supports data property checking and test selection. Tea already provides initial support for such interaction through its two modes of treating user assumptions. Human-in-the-loop systems could go back and forth between strict and relaxed adherence to user assumptions depending on sample size, data property, and user expertise. Scaffolding this back and forth process with visualization to enable novice analysts to learn statistical analysis skills is an interesting avenue for future research.

Expanding beyond NHST. Tea’s architecture is designed to be flexible and support extension. Currently, Tea provides a module for NHST because NHST is the most common paradigm in HCI. As statistics norms change, it will be important for Tea to support a broader range of analyses, including regression and Bayesian inference.

Extending Tea’s architecture and language to Bayesian inference presents several research challenges: (1) easing the process of choosing and expressing priors, (2) easing the process of choosing and expressing models, and (3) suggesting appropriate statistical tests. A variety of probabilistic programming languages emphasize language abstractions that let programmers succinctly express priors and models—BUGS [35], BLOG [38], Stan [7], Church [18], and Figaro [42] are a few prominent examples. Existing work suggests appropriate statistical tests for a researcher’s goals [30, 31, 37], but these suggestions are not embodied in a tool, language, or programming environment; we look forward to developing ways to encode these into Tea.

DISCUSSION
This paper introduces Tea, a high-level programming language that supports users in formalizing and automating statistical analysis.

Towards Task-Appropriate Analyses. Our evaluation shows that Tea’s constraint-based system to find suitable statistical tests generally matches the choices of experts. In particular, it automatically switches to non-parametric tests when parametric assumptions are not met. When preconditions are not met, Tea will always default to tests with fewer assumptions about the data, all the way to the bootstrap [12]. Tea prevents conducting statistical analyses that rely on unfounded assumptions. Given Tea’s automated test selection and assumption checking, analyses are more likely to be sound than is currently the case [6].

Towards Reproducible Analyses. Researchers have suggested automation as an opportunity to increase the transparency and reproducibility of scientific experiments and findings [43]. Tea programs are relatively straightforward to write and read and therefore could serve as a way for researchers to share their analysis for others to reproduce and to extend.

Towards Trustworthy Analyses: Pre-registration. Pre-registration holds the promise of promoting trustworthy analyses—e.g., by eliminating HARKing [28] [8], “p-hacking”, and “cherry picking” results. Tea can amplify ongoing efforts for pre-registration by providing a standard format for expressing study designs, hypotheses, and researcher assumptions.

Even before collecting data, researchers can write Tea programs to explicitly state their experimental designs, assumptions, and hypotheses. Without data, Tea will (i) check that the program is syntactically correct and (ii) check for consistency between variable declarations and hypotheses to ensure that hypotheses are well-formed. If the assumptions hold with the
Tea extends prior work on domain-specific languages for the data life cycle, tools for statistical analysis, and constraint-based approaches in HCI.

**Domain-specific Languages for the Data Life Cycle**

Prior domain-specific languages (DSLs) have focused on several different stages of data exploration, experiment design, and data cleaning to shift the burden of accurate processing from users to systems. To support data exploration, Vega-lite [44] is a high-level declarative language that supports users in developing interactive data visualizations without writing functional reactive components. PlanOut [3] is a DSL for expressing and coordinating online field experiments. More niche than PlanOut, Touchstone2 provides the Touchstone Language for specifying condition randomization in experiments (e.g., Latin Squares) [14], essential aspect of the domain knowledge users encode in Tea programs. To support rapid data cleaning, Wrangler [25] combines a mixed-initiative interface with a declarative transformation language. Tea can be integrated with tools such as Wrangler that produce cleaned CSV files ready for analysis.

In comparison to these previous DSLs, Tea provides a language to support another crucial step in the data life cycle: statistical analysis.

**Tools for Statistical Analysis**

Research has also introduced tools to support statistical analysis in diverse domains. ExperiScope [19] supports users in analyzing complex data logs for interaction techniques. ExperiScope surfaces patterns in the data that would be difficult to detect manually and enables researchers to collect noisier data in the wild that have greater external validity. Touchstone [36] is a comprehensive tool that supports the design and launch of online experiments. Touchstone provides suggestions for data analysis based on experimental design. Touchstone2 [14] builds upon Touchstone and provides more extensive guidance for evaluating the impact of experimental design on statistical power. Statsplorer [50] is an educational web application for novices learning about statistics. While more focused on visualizing various alternatives for statistical tests, Statsplorer also automates test selection (for a limited number of statistical tests and by executing simple switch statements) and the checking of assumptions (though it is currently limited to tests of normality and equal variance). [50] found that Statsplorer helps HCI students perform better in a subsequent statistics lecture.

In comparison to Statsplorer, Tea is specifically designed to integrate into existing workflows. Tea can be executed in any Python environment, including notebooks, which are widely used in data analysis. Tea enables reproducing and extending analyses by being script-based, and the analyses are focused on hypotheses that analysts specify.

**Constraint-based Systems in HCI**

Languages provide semantic structure and meaning that can be reasoned about automatically. For domains with well-defined goals, constraint solvers can be a promising technique. Some of the previous constraint-based systems in HCI have been Draco [39] and SetCoLa [21], which formalize visualization constraints for graphs. Whereas SetCoLa is specifically focused on graph layout, Draco formalizes visualization best practices as logical constraints to synthesize new visualizations. The knowledge base can grow and support new design recommendations with additional constraints.

Another constraint-based system is Scout [46], a mixed-initiative system that supports interface designers in rapid prototyping. Designers specify high-level constraints based on design concepts (e.g., a profile picture should be more emphasized than the name), and Scout synthesizes novel interfaces. Scout also uses Z3’s theories of booleans and integer linear arithmetic.

We extend this prior work by providing the first constraint-based system for statistical analysis.
CONCLUSION
Tea is a high-level domain-specific language and runtime system that automates statistical test selection and execution. Tea achieves these by applying techniques and ideas from human-computer interaction, programming languages, and software engineering to statistical analysis. Our hope is that Tea opens up possibilities for new statistical analysis tools, helps researchers in diverse fields, and resolves a century-old question: “Which test should I use to test my hypothesis?”

USING TEA
Tea is an open-source Python package that users can download using Pip, a Python package manager. Tea can be used in iPython notebooks. Information on how to download Tea and the source code can be accessed at http://tea-lang.org.

ACKNOWLEDGMENTS
We are grateful to our anonymous reviewers and our shepherd, Rob Miller. We would also like to thank Emina Torlak, Chenguang Wang, and James Bornholt for feedback on the runtime system’s implementation; Dominik Moritz, Pavel Panchekha, and Martin Kellogg for reading early drafts of this paper; Jeffrey Heer, UW Interactive Data Lab, and UW Database Group for conversations and feedback. This work is supported by a National Science Foundation (NSF) Graduate Research Fellowship, NSF grant #1651487, and NSF grant #1617892.

REFERENCES


