# Introductory statistics with `intRo`

Eric Hare
Iowa State University
and
Andee Kaplan
Iowa State University

**Abstract**

`intRo` is a web-based application for performing basic data analysis and statistical routines. Leveraging the power of R and Shiny, `intRo` implements common statistical functions in an extensible modular structure, while including a point-and-click interface for the novice statistician. This simplicity lends itself to a natural presentation in an introductory statistics course as a substitute for other commonly used statistical software packages, such as Excel and JMP. `intRo` is currently deployed at the URL http://www.intro-stats.com. Within this paper, we introduce the application and explore the design decisions underlying `intRo`, as well as highlight some challenges and advantages of reactive programming.

*Keywords:* Interactivity, Programming Paradigms, Reactive Programming, Reproducibility, Statistical Software

# 1 Introduction

The widespread adoption of R as a tool for statistical analysis has undoubtedly been an important development for the scientific community. However, using R in most cases still requires a basic knowledge of programming concepts which may pose a steep learning curve for the introductory statistics student. This additional time commitment may explain why introductory courses often utilize software applications such as JMP, even though instructors tend to use a tool like R in their own work. Still, some compromises must be made when using software like JMP, including dealing with software licenses and unsupported desktop platforms.

In teaching Introduction to Business Statistics at Iowa State University, we witnessed profound struggles by students attempting to practice introductory concepts discussed in class using current software. Scrimshaw (2001) notes in his manuscript that "open-ended packages, like any others, may create obstacles to learning simply through their lack of user-friendliness in the sheer mechanics of operating them, rather than any intrinsic difficulty in the content. . . ." In our own experience teaching, students' struggles were often directly related to the use of the software and not any sort of fundamental misunderstanding of the material, in agreement with Scrimshaw's finding.

Multiple software packages have recently been written in an attempt to spur student interest in introductory statistics and R programming. DataCamp's (DataCamp 2014) courses are user-friendly ways to learning basic R programming and data analysis techniques. Swirl (Carchedi et al. 2014) is a similar interactive tool to make learning R more fun by learning it within R itself. Project MOSAIC (Pruim, Kaplan, and Horton 2014) has created a suite of tools to simplify the teaching of statistics in the form of an R package. The primary goal of DataCamp and Swirl is to teach R programming, rather than facilitate the learning of introductory statistics. Project MOSAIC's goal is to faciliate this learning, but using the package requires a knowledge of R programming that the introductory student may not have.

Upon the release of RStudio's Shiny (RStudio and Inc. 2014) it became easier for an R-based analysis to be converted to an interactive web application. This in turn led us

to create an introductory statistics application which we call `intRo`, available at http://www.intro-stats.com. `intRo` offers a number of key advantages over traditional statistics software, including ease of use and an aim to foster student interest in coding. Additionally, `intRo` stands apart from new tools in that it is a supplement to an existing class, fully usable by a beginning statistics student. These advantages will be discussed at length in this text. The paper is structured as follows: section 2 introduces the application, its features and its usability. Section 3 describes the process of creating a classroom instance of `intRo` tailored to the individual instructor. Section 4 provides technical details on the implementation of `intRo`. Finally, section 5 discusses some future possibilities and limitations of the software.

## 2 What is `intRo`?

`intRo` is a web-based tool to accompany an introductory statistics class. It is meant to assist in the learning of statistics rather than as a stand-alone delivery of statistics education, with the intention of being used in conjunction with a guided class. An accompanying R package, titled `intRo` and available on GitHub, assists in the downloading, running, and deploying of `intRo` instances.

Three fundamental philosophies that guided the creation of `intRo`. In particular, `intRo` is *easy* to use and can be an *exciting* part of learning statistics. Additionally, `intRo` is an *extensible* tool, allowing for a user of `intRo` to tailor the tool for his or her own classroom needs.

In the development of `intRo`, we focused on aspects of the user interface (UI) and output that make it easy to pick up without extensive training. We used large, easy to click icons in the page header to help students find what they need more easily. We also made the functionality available the minimal necessary for an introductory statistics course. Figure 1 illustrates the simple steps a student takes to generate a result in `intRo`. In this instance, a user clicks on the Graphical tab to create a mosaic plot. The user sees the plot, and elects to click the save button to store the plot (and its corresponding code) to the final compendium.

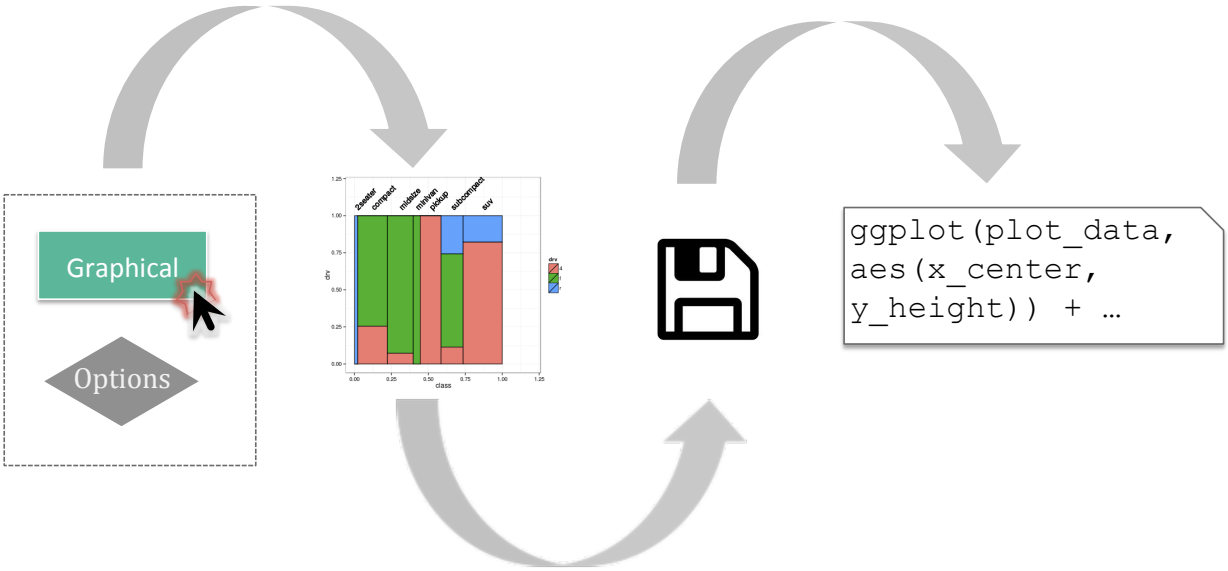


Figure 1: A typical user experience of generating a result in 'intRo'. In this instance, a user clicks on the Graphical tab to create a mosaic plot. The user sees the plot, and elects to click the save button to store the plot (and its corresponding code) to the final compendium.

Beyond being simple, `intRo` is also consistent. The tool is organized around specific tasks a student may perform in the process of a data analysis, called modules. To the user, a module is simply a page of statistics functionality that maintains a consistent layout, helping the user to become familiar with the location of the options, the results, and the code. Figure 2 highlights the five elements that comprise the `intRo` interface.

1. **Top Navigation** - The top navigation bar includes two sets of clickable icons. The left-aligned buttons are informational buttons. The first is a link to `intRo`. The second is a link to the documentation page. The third is a link to the GitHub repository where the code for `intRo` is housed. The final button is a link to our websites, which contain contact information if there are any questions or comments. The right-aligned buttons are `intRo` utilities. The first is a link to toggle the visibility of the code panel (5). The middle icon downloads an rmarkdown (Allaire et al. 2014) document of the analysis performed. The last is a link to print the stored module results, and the associated code (if visible).

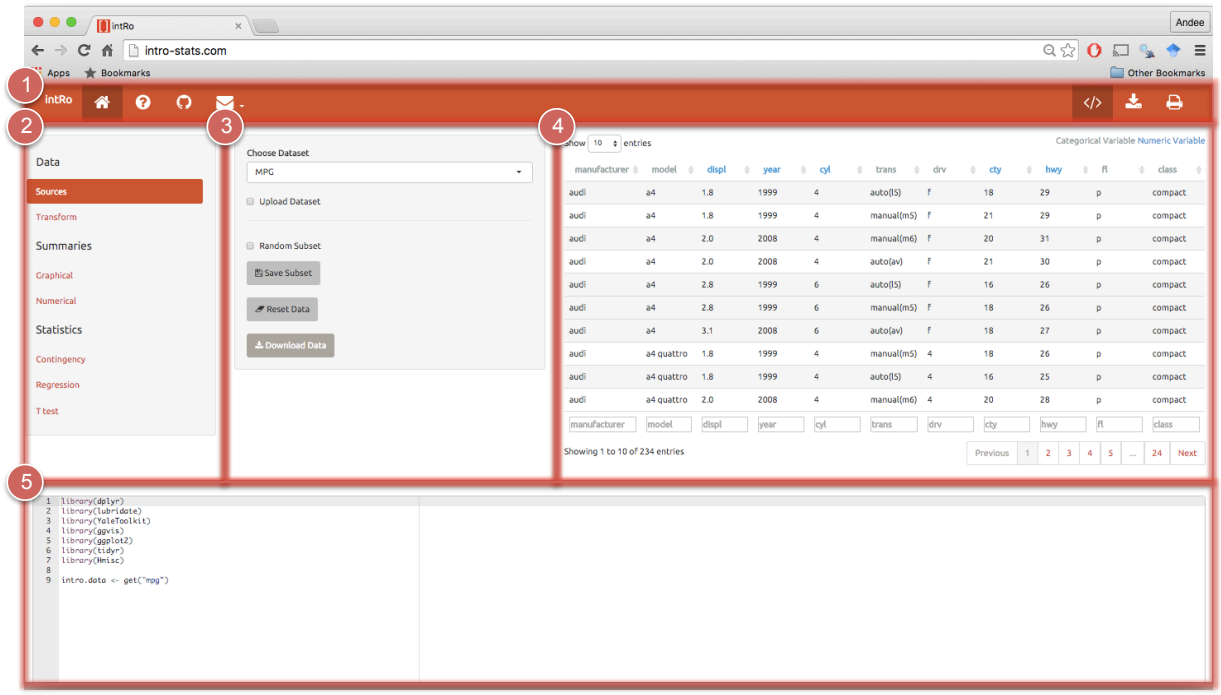


Figure 2: The five elements that comprise the 'intRo' application: 1) top navigation, 2) side navigation, 3) options panel, 4) results panel, and 5) code panel.

2. **Side Navigation** - The side navigation panel includes a list of data analysis tasks.
3. **Options Panel** - The options panel includes task-specific options which the user can use to customize their results.
4. **Results Panel** - The results pane displays the result of the selected module and options.
5. **Code Panel** - The code panel displays the R code used to generate the results from the student's `intRo` session. The code panel is shown by default to facilitate a transition to coding, but can be hidden by clicking the code toggle button in the Top Navigation bar.

The modules included in `intRo` are split into three higher level categories - data, summaries, and statistics. Under each of these categories, there are seven default modules, which perform specific data analysis tasks that employ an easy to use point-and-click interface. More modules can easily be added by an instructor, as detailed in section 4.1. The default

modules support uploading and downloading a dataset, transforming variables, graphical and numerical summaries, simple linear regression, contingency tables, and T-tests.

We've also created a documentation website that is consistent with the interface of the application. The documentation is available at http://gammarama.github.io/intRo and hyperlinked within the application itself.

`intRo` has an ulterior motive as well: to get students excited about programming. By navigating about the user interface of `intRo`, students are actually creating a fully-executable R script that they can download and run locally as well as viewing the script change real-time within the application. This code creation element of `intRo` is meant to generate excitement about programming in R and empower students to feel that they can generate code as well. `intRo` uses rmarkdown's render function in order to print the results, by dymanically executing the user's R script. By default, the output will include the R code, but if the user elects to hide the source code by clicking the code toggle button at the top, the code will not appear in the printed results.

On the front end, user interaction with `intRo` is split into bitesize chunks that we call modules. Modules are self-contained pieces of functionality which implement common statistical procedures that are often used in introductory statistics classes.

Within the data category there are two modules, sources and transformation. The data sources module allows the user to select or upload a dataset. Several datasets are pre-included, which can be selected via a drop-down menu. The user can check a box to upload a dataset, and then select the CSV file from her computer. Upon selection or successful upload of a dataset, a table view of the data will be visible on the right. From this module the user can select a subset of the data either by value or random selection. The dataset selected in the data module will be made available to all other modules of `intRo`. The data transformation module allows transformation of numeric variables to categorical and categorical variable to numeric. Additionally, numeric variable can be transformed using a power transformation. The transformed variables can be saved to the dataset for use elsewhere in `intRo`.

The graphical summaries module provides several common graphical displays of variables

in the dataset. First, the user selects the plot desired, and either a single variable or two variables from the dataset, depending on plot type. The variable available to choose will only be those compatible with plot type selected. The plot visible on the right will automatically update when these values are selected. The numeric summaries module is another method of computing summaries of the data. The user is presented with a list of each variable from their dataset. The user can select one or more different variables, and will be presented with a table of some summary statistics. The summary statistics displayed vary depending on the type of the variables. For instance, for a numeric variable, a five number summary will be displayed. Character or factor variables will display counts of each unique value of that particular variable. Additionally, the user has the ability to display summaries by a grouping variable.

Within the set of statistics modules, there are three simple statistics tasks that an introductory statistics student will learn in her first semester. The contingency table module allows the user to compute a contingency table for two categorical variables. The results can either be displayed as counts, row percentages, column percentages, or total percentages. The simple linear regression module allows the user to compute a simple linear regression of a dependent variable (y) on an independent variable (x). The module will automatically filter any variables that are non-numeric so that the user may only select variables that are suitable for a simple linear regression. Results of the regression will be displayed on the right. A table of parameter estimates, a scatterplot of the data along with a line of best fit, The R-Squared value, and residual diagnostic plots will all be displayed. A user also has the ability to save residuals and predicted values to their dataset for further analysis. The T test module allows the user to perform either a one variable or two variable Student's T test. The user can select configuration options, including the direction of the test and the confidence level of the test. The results, including the t statistic, degrees of freedom, p-value, and a confidence interval, will be displayed in a text box in the results panel.

Each module is a self contained set of R code that is dynamically added to the application at run time. What this means is that `intRo` can be easily extended by the addition of modules within the framework underlying the application. This extensibility allows for `intRo` to be customized for an individual classroom setting, tailored to the needs of a particular course.

As an example of possible customization, an instructor may wish to introduce hypothesis testing from a nonparametric framework rather than distributional. In this case, a wilcoxon rank sum test could be added easily as a module to be used instead of the T test. This addition would be fairly simple to complete. The user would fork `intRo` from http://github.com/gammarama/intRo and begin by adding six types of files to a folder within a module folder: `helper.R`, `libraries.R`, `observe.R`, `output.R`, `reactive.R`, and `ui.R`. Each of these files is written to comply with the reactive nature available in a Shiny application. Further details on the creation of new modules can be found in section 4.1.

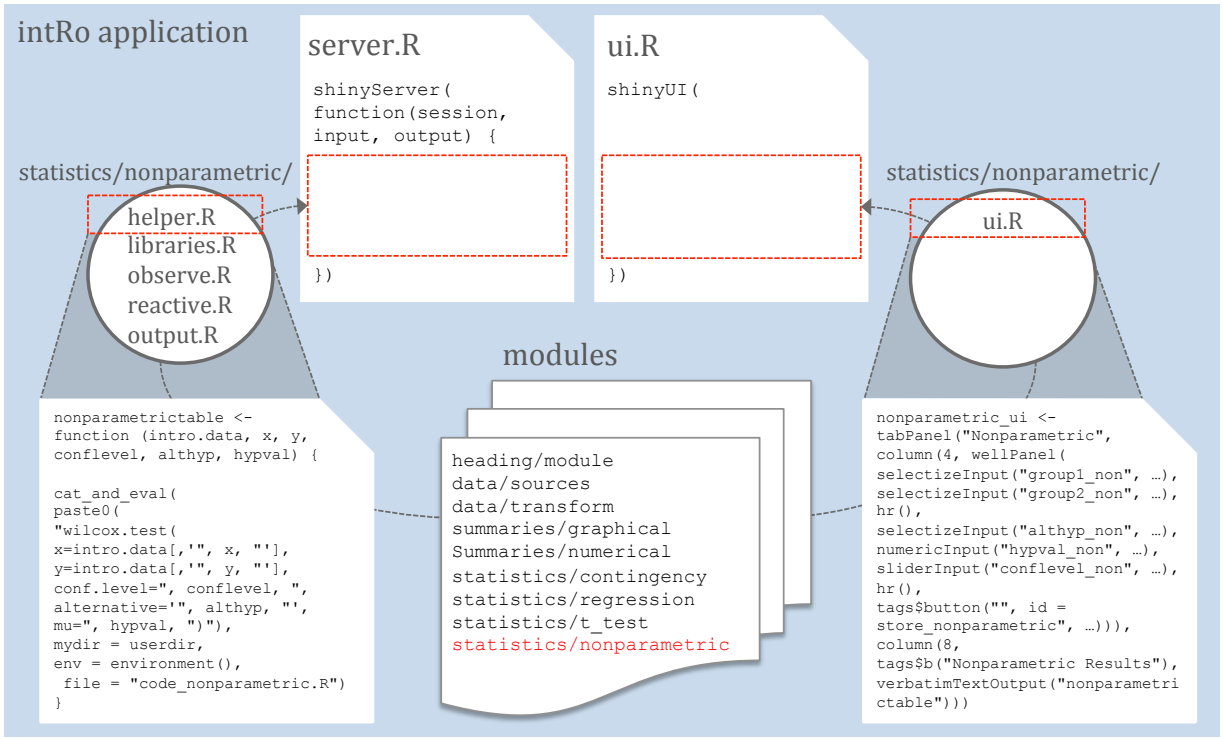


Figure 3: This figure depicts how the `server.R` and `ui.R` files are populated using the modular structure within 'intRo'.

The process by which `intRo` is dynamically created upon running the application through module sourcing is detailed in figure 3. The key driver to populating `server.R` and `ui.R` is the navigation file `modules.txt`, which defines the placement of each module. The interface is then created with the following statement.

```
## Source ui
module_info <- read.table("modules/modules.txt",
    header = TRUE, sep = ",")
sapply(file.path("modules",
    dir("modules")[dir("modules") !=
        "modules.txt"], "ui.R"),
    source)

## mylist is a list
## containing the different
## ui module code Create
## the UI
shinyUI(navbarPage("intRo",
    id = "top-nav", theme = "bootstrap.min.css",
    tabPanel(title = "", icon = icon("home"),
        fluidRow(do.call(navlistPanel,
            c(list(id = "side-nav",
                widths = c(2,
                  10)), mylist)))),
    ...))
```

The key piece of code being the `do.call` statement loading the list of ui elements from the module's `ui.R` file. The server functions are then dynamically generated using a similar method.

```
shinyServer(function(input,
    output, session) {
    ## Module info
    module_info <- read.table("modules/modules.txt",
        header = TRUE, sep = ",")

    ## Modules
    types <- c("helper.R",
        "static.R", "observe.R",
        "reactive.R", "output.R")
    modules_tosource <- file.path("modules",
```

```
        apply(expand.grid(module_info$module,
            types), 1, paste,
            collapse = "/"))

    ## Source the modules
    for (mod in modules_tosource) {
        source(mod, local = TRUE)
    }
})
```

In this way, we were able to have `intRo` be fully extensible, its structure and functionality dependent entirely on the modules present within the application.

# 3 Using `intRo`

Although users can access intRo from http://www.intro-stats.com, course instructors may wish to customize their own instance.

## 3.1 `intRo` package

`intRo` can be downloaded, ran, and deployed on ShinyApps.io through the use of the R package `intRo`. Currently, the package is only available on GitHub, and can be installed using the devtools package as follows:

```
devtools::install_github("gammarama/intRo")
```

After installing the `intRo` package, the first function one should call is `download_intRo`. `download_intRo` takes as an argument a directory in which to store the application. By default, it selects the working directory of the R session. This function clones the application branch of the `intRo` repository on GitHub, and hence will pull the latest version of the code whenever it is ran.

Running `download_intRo` will produce an `intRo` folder in the specified folder. It can then be ran as any Shiny application, using Shiny's `runApp` command. However, we have provided a wrapper function `run_intRo` which adds some additional customization options to the execution process. `run_intRo` takes as argument the path to the folder containing the `intRo` application. It also takes several more optional arguments:

- `enabled_modules`: A character vector containing the modules to enable
- `theme`: A string representing a shinythemes theme to use
- `...`: Additional arguments passed to Shiny's `runApp` function

The package provides help documentation which explains in further detail the format that these arguments would take, but as an example, suppose I wanted to download `intRo` to my working directory, execute an `intRo` session with only the data sources, data transform, and numerical summaries modules enabled, and apply the cerulean theme. The series of calls to do so would be as follows:

```
download_intRo()
run_intRo(enabled_modules = c("data/transform", "summaries/numerical"),
          theme = "cerulean")
```

Note that the data sources module is required, and hence must be included in all intRo sessions and need not be specified in the enabled_modules argument.

If the intent is to use a specific instance of `intRo` where many users will access it at the same time, such as in an introductory statistics class, it may be preferable to deploy a custom instance of `intRo` to a publicly accessible URL. The package provides a function `deploy_intRo` which is a wrapper for the `deployApp` function contained in the shinyapps package. Once the shinyapps package is installed and configured, `deploy_intRo` will upload `intRo` as an application on the user's account. The function takes the same arguments as `run_intRo`, so it can be deployed with a custom selection of modules, and a customized theme. It also takes an additional argument `google_analytics`, which allows the specification of a Google Analytics tracking ID. It also takes `...` as additional arguments to be passed into

the `deployApp` routine. For example, if we wished to deploy the instance of `intRo` we ran previously, we would call it like so:

```
deploy_intRo(enabled_modules = c("data/transform", "summaries/numerical"),
             theme = "cerulean")
```

Once the process finished, the app will become available at http://<user>.shinyapps.io/intRo, where `<user>` is the username of the ShinyApps.io account configured.

## 3.2 In the classroom

`intRo`'s ease of use lends itself to a natural presentation in an introductory statistics course. Statistics software typically used in introductory courses, such as JMP and Excel, have restrictive software licensing and platform requirements that can limit the software's accessibility to students. Even if these limitations can be overcome, these software programs have non-ideal interfaces for an exploratory data analysis. JMP, for instance, hides several basic regression functions such as a line of best fit behind small red arrows. Using R itself in the classroom is an appealing alternative, but suffers from some other problems. Many students in introductory statistics courses have not been exposed to programming languages and concepts, which could require a substantial overhead in order to begin a basic statistical analysis. When results need to be submitted as homework, getting the output into a document can be a frustrating and tedious experience (this experience is made much easier with new tools like rmarkdown, but this again is a bit of a learning curve for the new statistics student).

We believe that `intRo` targets these key limitations. The interface is designed to be quick, seamless, and easy to understand. Because `intRo` is web-based, the statistical analysis can be performed from nearly any modern device. R underlies the analysis, but users need not be familiar with R or programming to perform it. The focal point of the interface is the output panel, allowing the student to remain focused on the results and limiting the need to fight with the interface itself to get work done. For those students who may have a natural

inclination for programming, executable code is generated to allow a local reproduction of the analysis.

The features of `intRo` were selected in order to apply to the most commonly used lesson plans in early statistics courses. The textbook "The Art and Science of Learning from Data" by Agresti and Franklin (2012) references many of the focal points of introductory statistics, and `intRo` supports performing a majority of them. Part one, titled "Gathering and Exploring Data", covers graphical summaries, measures of spread and center, and avoiding the misuse of graphics in situations they are not appropriate. `intRo` includes modules for both graphical and numerical summaries covering these topics. Furthermore, `intRo` limits the display of variables to ones that are appropriate for a chosen plot type, helping to reinforce the appropriateness of particular graphics (Table 1).

| Plot Type | X Variable Type Allowed | Y Variable Type Allowed |
|---|---|---|
| Histogram | Numeric | N/A |
| Normal Quantile Plot | Numeric | N/A |
| Scatterplot | Numeric | Numeric |
| Line Chart | Numeric | Numeric |
| Boxplot | Categorical | Numeric |
| Bar Chart | Categorical | Numeric |
| Pareto Chart | Categorical | Numeric |
| Mosaic Plot | Categorical | Categorical |

Table 1: The plot and variable types allowed in 'intRo'' s graphical module. For example, when the selected plottype is histogram, the interface displays only a single variable selection box (x), and the variables selectable in this box are only numeric. If a boxplot is selected, the x variable must be categorical and the y variable must be numeric, and the list of choices updates automatically to reflect this.

Part two of this text discusses probability and sampling distributions, which is outside the current realm of `intRo`, but part three focuses on inferential statistics. The inferential statistics include confidence intervals, hypothesis testing, and comparing two variables, all supported natively in `intRo`. Finally, part four of the text discusses the association between

variables, including regression, functions that `intRo` performs with ease. Some of the more advanced topics of this section are currently unsupported by `intRo`, including multiple regression.

Statistics courses with a lab component can most benefit from the use of `intRo`. As students learn the aforementioned concepts, they can practice using the tools to perform the statistical tests and summaries. They simply upload a dataset and click the module appropriate to the section of the class. Once they choose the options desired by the instructor (variables, confidence levels, etc.), they can view the results and ultimately print the output for turning in. Early labs can walk the user through the limited number of button presses and clicks necessary, while later labs can more vaguely describe the intended output in hopes that the user can easily navigate the `intRo` interface to obtain the results. For topics currently unsupported by `intRo`, the instructor can supplement its use with other web-based modules that are publicly available or create their own modules within `intRo` as detailed in section 4.1.

# 4 Understanding `intRo`

To fully understand `intRo`, it is important to expand upon three underlying concepts. The first is the idea of modularity in the context of a Shiny application. The second is the idea of reproducibility in the context of a graphical user interface. Finally, the third is the idea of reactive programming in the context of a web application.

## 4.1 Modules

An `intRo` module is a set of self-contained executible R scripts that together produce a set of introductory statistics functionality. `intRo` modules were designed in this way to allow for simple dynamic creation of the user interface at run-time, as well as ease the process of converting existing analysis code to the `intRo` framework. In this section, we detail the structure of these modules, and the process of creating and deploying new modules.

A module consists of the following scripts:

- *helper.R* - R code that performs some statistical analysis or transformation. This would typically be in the form of a function, and similar to any standard R script.
- *libraries.R* - Code to load any libraries which are not part of core R.
- *observe.R* - Shiny observer code typically used to update choices of an input box.
- *output.R* - Shiny output code defining the results of the analysis that should be displayed to the user.
- *reactive.R* - Shiny reactives, typically containing data that depend on inputs.
- *ui.R* - Shiny user interface definition, including the placement of the inputs and outputs.

The modules provided with `intRo` are contained in the modules folder. The top level directory in the modules folder defines the category of the module (currently `data`, `summaries`, or `statistics`). Within each of these categories is a folder named according to the name of the module. This folder houses the previously defined scripts. As an example, we will walk through the contents of the `contingency` module.

Since the `contingency` module performs a statistical analysis, it is part of the statistics category, and hence can be found in the `intRo` repository at **modules/statistics/contingency**. Let's first examine *helper.R*:

```
cont.table <- function(intro.data, x, y, type, digits) {
    interpolate(
        ~ (my.tbl <- cbind(rbind(table(df$y, df$x),
                      Total = colSums(table(df$y, df$x))),
                Total = c(rowSums(table(df$y, df$x))[-(length(df$x))],
                          sum(table(df$y, df$x))))),
        df = quote(intro.data),
        x = x,
        y = y,
        mydir = userdir,
        `_env` = environment(),
        file = "code_contingency.R"
```

```
    )

    # Additional formatting code suppressed for readability
    ...

    interpolate(~(format(round(my.tbl, digits = new.digits))),
                mydir = userdir,
                `_env` = environment(),
                file = "code_contingency.R",
                append = TRUE)
}
```

This script is most immediately similar to standard R code. In this case, a function `cont.table` is created which, depending on the values of the parameters, ultimately returns a contingency table. One important difference from a typical R script is that each call in the script is wrapped in a function called `interpolate`. `interpolate` both executes the given R code on the server, and also writes the code executed to the script window at the bottom of `intRo`. More details on the reasons and choices behind `interpolate` can be found in section 4.2.

Because all the code needed to implement a contingency table is found in the `base` and `stats` package, the `libraries.R` file is empty for the `contingency` module. `observe.R`, which defines the Shiny observers needed, is reproduced below:

```
observe({
    xselect <- ifelse(checkVariable(intro.data(), input$xcont),
                      input$xcont,
                      intro.categoricnames()[1])
    yselect <- ifelse(checkVariable(intro.data(), input$ycont),
                      input$ycont,
                      intro.categoricnames()[2])
```

```
    updateSelectizeInput(session,
                         "xcont",
                         choices = setdiff(intro.categoricnames(),
                                           yselect),
                         selected = xselect)
    updateSelectizeInput(session,
                         "ycont",
                         choices = setdiff(intro.categoricnames(),
                                           xselect),
                         selected = yselect)
})

observeEvent(input$store_contingency, {
    cat(paste0("\n\n", paste(readLines(
        file.path(userdir, "code_contingency.R")),
        collapse = "\n")),
        file = file.path(userdir, "code_All.R"),
        append = TRUE)
})
```

Shiny observers are a class of reactive objects within the Shiny paradigm which do not return a value (RStudio and Inc. 2014). For further discussion of reactivity, see section 4.3. In this example, observers are created to ensure that the choices of variable for the `contingency` module are only categorical variables. This is accomplished by utilizing the global reactive `intro.categoricnames()`, which returns a character vector containing the variables in the current dataset that are categorical. Finally, there is an event observer to store code generated from the module into the overall code script upon clicking the store button. The presence of this observer code and the definition of the button in the user interface are enforced, and must be present in any `intRo` module.

The `output.R` and `reactive.R` code is very simple:

```
output$conttable <- renderTable({
    return(intro.contingency())
})
```

```
intro.contingency <- reactive({
    if (is.null(intro.data())) return(NULL)
    if (input$xcont == input$ycont ||
        length(categoricNames(intro.data())) < 2) return(NULL)

    my.conttable <- cont.table(intro.data(),
                               input$xcont, input$ycont,
                               input$conttype, input$contdigits)

    return(my.conttable)
})
```

`intro.contingency` is defined as a reactive object. Some small error checks are performed at the beginning to ensure that the data is present and available, and that two different categorical variables are selected. Assuming those conditions hold, a call to the previously defined `helper.R` function is made with the appropriate parameters, and the results are returned. The `output.R` script then simply uses Shiny's `renderTable` function to display the resulting table.

Finally, the `ui.R` code is shown below:

```
contingency_ui <- tabPanel("Contingency",
    column(4,
         wellPanel(
             selectizeInput("xcont", label = "X Variable (x)",
                            choices = categoricNames(mpg),
                            selected = categoricNames(mpg)[5]),
```

```
            # Additional inputs suppressed for readability
            ...

            hr(),

            tags$button("", id = "store_contingency", type = "button",
                        class = "btn action-button",
                        list(icon("save"), "Store Contingency Result"),
                        onclick = "$('#top-nav a:has(> .fa-print,
                        .fa-code, .fa-download)').highlight();")
        )
    ),

    column(8,
        tableOutput("conttable")
    )
)
```

This script defines all the inputs and outputs that the user will see. The only requirements from `intRo`'s perspective are (1) that there exist a store button at the bottom of the middle panel for storing the results of the analysis in the code script, and (2) that configuration options appear in the width 4 column in the middle, and output appears in the width 8 column on the right. The remaining input and output definitions depend on the statistical analysis or transformation being performed.

Although the structure of an `intRo` module is relatively straightforward, producing the code needed in a more seamless fashion would certainly help open up the creation of such modules to a wider audience. As we discuss in the conclusions and future work section, providing an `intRo` module creation tool to abstract away some of the less common coding paradigms, like the use of `interpolate`, is an important effort that will continue to be pursued.

## 4.2 Reproducibility

Though they can make common actions quicker and interacting with software much simplier, many Graphical User Interfaces for scientific software packages come with a major drawback. That is, the actions taken by the user are typically not reproducible. Code scripting, on the other hand, allows us to share a file containing code, and another user can execute the code to reproduce the same results. In terms of the scientific process, where results can and should be scrutinized by other parties, the utility of GUIs are greatly limited.

We have designed `intRo` modules to reproduce actions taken by the user so that they can obtain R code associated with each module interacted with. When a user chooses to store the results associated with a particular module, R code yielding the visible results is automatically added to the code window at the bottom. This framework yields several advantages:

1. This eases a user who may be intimidated by programming into the idea that interacting with a user interface is really just a frontend for code. Seeing the correspondence between graphical clicks and printed code will hopefully also lessen the fear of coding that many inexperienced users may have.

2. An analysis created by `intRo` can be reproduced in an R session to easily assess and extend the results.

3. "Printing" the results of an `intRo` analysis amounts to nothing more than executing the R code on the server, adding another layer of reproducibility.

We designed the modules in such a way that the code printed to the console is the same as the code executed by the server, which makes the process of writing an `intRo` module significantly more seamless. The workhorse function for this reproducibility framework is called `interpolate`, which is based on a function proposed by Hadley Wickham (2015). The idea of interpolation is that by using non-standard evaluation, the call can be captured, printed, and evaluated, with all the appropriate arguments substituted into the call automatically. This means that depending on the options selected in `intRo`, the corresponding arguments to functions are appropriately set for reproduction in a standard R session.

## 4.3 Reactive design choices

Reactive programming is a programming paradigm that "tackles issues posed by event-driven applications by providing abstractions to express programs as reactions to external events and having the language automatically manage the flow of time (by conceptually supporting simultaneity), and data and computation dependencies." (Bainomugisha et al. 2012) As implemented by Shiny, results automatically update when users interact with the interface.

intRo leverages the reactive programming nature of Shiny, and as such is designed around the idea of user input cascading through the entire application. In a typical Shiny application, users interact with inputs that act as parameters to function, which in turn yield different results. Within `intRo`, the users are able to interact with and manipulate the data underlying the entire application. This posed many challenges in the creation of `intRo` and drove design decisions, namely timely save points according to the user's workflow, and reactive updating of variable lists tied to inputs across the entire application. Because the user may experiment with different configurations or select different variables, we did not want to store all actions taken in the intRo session. Rather, each module includes a button allowing the user to explicitly store the output visible in the results panel into the R script. This way, output is only stored when the user is satisfied, and the resulting output is not cluttered with unnecessary information.

In the creation of `intRo` we walked a fine line between giving the user flexibility and having realistic usability. At the same time, `intRo` was created as a consumer of another package, Shiny, in which we as developers were the beneficiaries of another team of developers' decision to balance flexibility and usability. For a tangible example, consider the graphical summaries module. We only allow variables of a type consistent with the selected plot to be displayed. This is a conscious decision that limits an `intRo` user's flexibility, while maximizing the usability (by minimizing crashes) of the application. On the flip side of this, Shiny allows much higher flexibility. For instance, the entire application (including user interface) is created dynamically upon load, based on the modules currently housed within `intRo`. However, Shiny does have limits on its flexibility based on the designers decisions for usability. One current example is the slider element. This element allows for fixed width

steps from its minimum to its maximum. The JavaScript library being utilized in Shiny allows for arbitrary function calls to to generate these steps, however they must be written in plain JavaScript. This is an example of a decision made by the developers of Shiny to limit functionality in favor of usability of their package.

# 5 Conclusions and future work

We believe `intRo` can be a powerful and effective tool for introductory statistics education. Its modular structure allows it to be flexible enough for many different applications and curriculums. Its ease-of-use allows the student to focus her attention on the statistics task at hand, rather than struggling with software licenses and confusing interface navigation. Reproducible code generated from each analysis can be used to spark an interest in R programming in those who might otherwise not be exposed to it.

In addition to the current functionality, there are some practical improvements in the works that will make `intRo` more useful to both students and instructors. In particular, we have begun development on an R package which will allow `intRo` modules to be created automatically from user written R code. This package will generate the necessary file structure to allow the module's incorporation into `intRo` as well as translate user code to `intRo` compatible code and populate the necessary files. This will vastly improve `intRo`'s flexibility and allow it to be used in a wider range of curricula, including more advanced statistics courses. Additionally, we would like to utilize the expanded interactive capabilities of ggvis in order to make `intRo's` plots more engaging to students. One way to do this would be implementing linked plots, in which interactions with one plot are reflected in other plots that illustrate the same data. This would be particularly useful in the regression module so that students could explore observations with high influence and leverage.

We hope to use `intRo` in courses to collect feedback regarding the ease of use and functionality. This will allow us to assess its usefulness relative to software used in the past, as well as gauge areas for improvement. Furthermore, we can determine the effectiveness of code printing on generating excitement from the students about programming in R.

Challenges do exist with regards to the wider adoption of `intRo`. For instance, we will need to monitor how well the server hosting `intRo` handles the load of dozens of students performing data analyses at once. If performance issues are encountered, the infrastructure used may need to be expanded to handle current and future load. An unknown quantity will be how feasible it is to increase adoption of `intRo` across the department, as well as to other universities. One limitation of `intRo` is that uploading a dataset beyond about 30,000 rows tends to be slow. Even once the data is successfully uploaded, the default modules produce results more slowly than with smaller datasets. This is a limitation that should be further investigated if and when `intRo` sees wider adoption.

Regardless, tools that focus on usability and extensibility in statistics education, such as `intRo`, are sure to encourage the next round of innovators to be interested and excited about statistical computing.

# 6 Supplementary material

All code and documents related to this manuscript are available at https://github.com/gammarama/intRo.

# References

Agresti, Alan, and Christine Franklin. 2012. *Statistics: The Art and Science of Learning from Data*. Pearson Higher Ed.

Allaire, JJ, Jonathan McPherson, Yihui Xie, Hadley Wickham, Joe Cheng, and Jeff Allen. 2014. *Rmarkdown: Dynamic Documents for R*. http://CRAN.R-project.org/package=rmarkdown.

Bainomugisha, Engineer, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx, and Wolfgang De Meuter. 2012. "A Survey on Reactive Programming." In *ACM Computing Surveys*. Citeseer.

Carchedi, Nick, Bill Bauer, Gina Grdina, and Sean Kross. 2014. *Swirl: Learn R, in R.*

http://CRAN.R-project.org/package=swirl.

DataCamp. 2014. "Online R Tutorials and Data Science Courses - DataCamp." https://www.datacamp.com/.

Pruim, Randall, Daniel Kaplan, and Nicholas Horton. 2014. *Mosaic: Project MOSAIC (Mosaic-Web.org) Statistics and Mathematics Teaching Utilities*. http://CRAN.R-project.org/package=mosaic.

RStudio, and Inc. 2014. *Shiny: Web Application Framework for R*. http://CRAN.R-project.org/package=shiny.

Scrimshaw, Peter. 2001. "Computers and the Teacher's Role." *Knowledge, Power and Learning*. London, Paul Chapman Publishing Ltd.

Wickham, Hadley. 2015. "Graphics & Computing Student Paper Winners @ JSM 2015." https://github.com/hadley/15-student-papers.