jsslogo.jpg

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

iNZight: A Graphical User Interface for Data Visualisation and Analysis through R

Tom Elliott
Victoria University of Wellington

Chris Wild
University of Auckland

Daniel Barnett
University of Auckland

Andrew Sporle
University of Auckland

Abstract

Visualisation, exploration, and analysis of data is often inaccessible to many due to high up-front costs of learning the necessary coding skills to get started. Graphical user interfaces (GUIs) have often been used to provide inexperienced users with access to these underlying, complex systems without the need for coding. Yet, with respect to R GUIs, many still require some degree of experience with R. **iNZight** is a GUI-based tool written in R that provides researchers and students with the opportunity to interact with and explore data without the need for code. The tool is designed to be easy to use, with intuitive controls and clever defaults. **iNZight** also provides more complex features for manipulation and analysis of data, and includes some code-writing capabilities for researchers to efficiently generate reproducible outputs, or as a pathway for newcomers to learn the basics of the R programming environment.

Keywords: GUI, statistical software, statistical education, R.

1. Introduction

The R programming environment (R Core Team 2020) is an open source statistical programming environment used throughout statistics and data science and backed by a repository of thousands of packages covering even the most unique of problems. Amongst these packages are several graphical user interfaces (GUIs) providing point-and-click methods for creating graphs, performing hypothesis tests, and a range of other statistical methods, with two prominent examples: R Commander (Fox 2005) and **Deducer** (Fellows 2012). R Commander includes a full interface which displays, writes, and runs R code, while **Deducer** extends the R console with additional menus to open window interfaces to access a range of methods. These tools allow users to perform known procedures using point-and-click interfaces without needing to remember function and argument names. This does, however, require that users not only install R and the necessary packages, but also know in advance what thay want to achieve, excluding those new to statistics who are yet to learn these technical terms.

A preferable approach is to let users choose the variables first, and let the software present a selection of applicable methods. **iNZight** uses this "variable-first" approach, presenting users with an exploration-focussed interface emphasising graphics, which are at the core of exploratative data analysis. After visually exploring their data, users can produce summary statistics and inference information with the click of a button—**iNZight** simply presents information relevant to the chosen variables. For hypothesis testing, users choose from a list of tests applicable to the variables displayed in the graph without having to recall specific names. This not only makes **iNZight** ideal for users new to statistics and data science, but also as a rapid research development tool for organisations performing repetitive data analysis tasks that might otherwise require using tools that require special training or hiring specialised programmers. [1]

Like other GUIs, there is a code component. R Commander provides a place for users to enter R code, while **Deducer** sits on top of the R console using the environment as-is. With **iNZight**, however, code is evaluated "behind the scenes", and is not directly editable by users. Every

[1] Better wording.

action the user makes calls one or more R functions, and the code is added to the *code history* for users to review, save, and share. Users can generate an R script unique to their data and later edit and run the code manually in R, quickly generating a reproducible methodology for research oganisations. Additionally, the script can be used as a stepping stone for learning to code in R. [2]

Not only has **iNZight** been adopted throughout New Zealand's statistical education program, the combination of **iNZight**'s simplicity and powerful tool set make it a popular choice for research (and other)[3] organisations. Students are introduced to basic statistical concepts using **iNZight** in their final year of high school, and thus as future researchers will be familiar with it for their professional projects. This paper provides an overview of **iNZight**'s main features, technical details of its development, an introduction to the *add-on* system, and a description of the install process.

2. A tour of iNZight's features

iNZight was originally designed as a tool for teaching statistics to students, but is equally suited to research groups as an in-house tool for rapid research development. The intuitive interface, shown in Figure 1, makes **iNZight** very easy to learn and, unlike more complicated alternatives, can be picked up after a period of non-use with little effort, making it ideal for infrequent but critical tasks within organisations. The interface uses familiar controls such as *drag-and-drop*, *drop-down* selection, and *slider bars*. **iNZight** uses a *variable-first* approach, where users choose the variables they are interested in and let the software display the relevant actions. The best way to explore **iNZight**'s features is by demonstration.

2.1. Loading data

Data comes in a wide range of formats, some of which are typically software-dependent (for example Excel files, Microsoft Corporation 2018). Due in part to being open source, there are 1000's of R packages on the Comprehensive R Archive Network (CRAN), amongst which are

[2] Drop this paragraph in favour of one for student vs research?
[3] some examples from Andrew?

`figure/inzight_main.png`

Figure 1: The **iNZight** GUI landing page presents users with a few controls. The labeled areas are: (A) the active data set is displayed prominently; (B) variable control boxes provide users either drag-and-drop from (A), or select from dropdowns; (C) graphs are displayed in the graphics window; (D) plot controls, most importantly the plot configuration controls (left); (E) if enabled, code for the active plot is shown here and can be edited by the user.

figure/inzight_load_data.png

Figure 2: Load Data window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

some dedicated to reading specific file formats. However, users must still know the correct package and function for their files, and argument names and order. **iNZight** provides a simple LOAD DATA window from which users can select a file to import, at which point the software detects the file type from the file extension and attempts to read the file. If successful, a preview is displayed for the user to check before proceeding with the import. Figure 2 shows the LOAD DATA window, which has detected a comma separated values (CSV) file and used the appropriate function in the background. Currently, **iNZight** supports files in CSV, tab-delimited, Excel, SAS, Stata, SPSS, R-data, and JSON formats.

In addition to the preview, **iNZight** also has an ADVANCED OPTIONS section for some specific formats. Currently only delimited files have advanced options, where users can override the default delimiter, for example in European countries where the semi-colon (;) is used (the comma is reserved as the decimal separator), or to choose between different encoding formats. The preview is updated when these options are changed, so users do not have to know specifically what they need, and can use trial-and-error to get the preview looking right before importing the data into **iNZight**. This is particularly useful for encoding, which is difficult to find out manually.

Metadata for delimited files

One issue with delimited data formats (for example CSV) is lack of coding information about variables. For example, a vector with values 1, 2, 3, and 4 might be an integer, however it may also be a coded *factor* variable. Typically, users would have to convert this variable to a factor ("categorical" in **iNZight**) and then label the levels, which would be found in additional documentation supplied with the data. However, this is a lot to ask of novice users, and when there are 10s or even 100s of coded factor variables in a dataset, it becomes very tedious in a GUI system.

To work around this problem, we have developed a *metadata* system for supplying and implementing additional information. Currently, this comes in a very naïve format where information is added to the top of a file, but future endeavours will allow **iNZight** to read metadata from a separate file. As an example, say a variable called `var1` contains coded levels 1–4, with labels 'a', 'b', 'c', and 'd'. The metadata line below is added to the top of the CSV file:

```
#' @factor var1[a,b,c,d]
```

When the file is read in, **iNZight** automatically parses this metadata and imports the variable called `var1` as a factor with the appropriate labels.

The metadata support goes further, allowing renaming of levels, variable names, and even supplying missing-value codes in numeric-variables. The latter is common in surveys, in which 'missing value' responses may be coded with values such as 88 for "Don't know" and 99 for "Refused to answer". By providing this information in the metadata, users avoid producing graphs and summaries of the numeric variable including the coding missing values in the results.

2.2. Creating graphs

Within **iNZight**, graphics are at the core of the user experience: the first prototype included a `figure/inzight_infant_mortality.png`

Figure 3: Demonstration of plot modifcations available from **iNZight**'s Add to Plot menu. The Add to Plot button, highlighted in red, opens a panel giving user control over colours, size, shape, labels, and much more. This can also be accessed from the plot menu, boxed in blue.

drag-and-drop of variables to create a plot, and that is how things have remained. Behind the scenes, **iNZight** detects the variable types (numeric, categorical, or a date-time) and draws an appropriate graph. For example, a single numeric variable produces a dot plot, while a categorical variable produces a bar chart. The user does not need to know what type of graph they want to create from a chosen variable, allowing them to freely explore the dataset by choosing variables to look at.

The first variable box is the variable users are most interested in, and may want to explore how it is influenced by other variables. If *height* is chosen as Variable 1, we may want to see how height changes with ethnicity or age. In the first case, a set of dot plots of *height* will be displayed for each *ethnicity*, while in the latter case, the result will be a scatter plot with *age* on the x-axis (explanatory variable) and *height* on the y-axis (the response). In addition to the first two variable slots, **iNZight** includes two subset variables to quickly and easily facet the plot and explore more complex relationships and interactions.

For more in-depth exploration, there is an entire panel dedicated to plot modifications: Add to Plot. This is accessed either from the Plot menu, or from the button in the Plot Toolbar (boxed in red in Figure 3). From here, users can choose from a selection of alternative plot types (limited by the types of variables), as well as choose a colour variable, sizing variable, plot symbols, trend lines, changing axis labels and limits, and much more. The options are presented in an interactive format such that the graph updates whenever the user changes input values, allowing them to explore "what happens if . . . ", and "what does this do?". Beginners can explore a dataset without being limited by a lack of knowledge and coding skill, while researchers can quickly generate visualisations before starting their analysis. Figure 3 shows a graph produced by **iNZight** exploring the relationship between infant mortality and GDP, region (colour), population (point size), and year (faceting).

2.3. Summaries and inference

To supplement the visual graphics, **iNZight** provides two textual output modes: *summaries* and *inference*, accessed from the Get Summary and Get Inference buttons, respectively, at the bottom-left of the interface. Summary information includes basic information about the chosen variables, including mean, standard deviation, and quantiles, acting as a quick reference for values that are likely estimable from the graph itself. Often researchers are interested in a table of counts, which can be accessed here.

The inference information provides estimates, confidence intervals, and any applicable p-values for quantities such as means and proportions, and a simple interface for performing hypothesis tests. **iNZight** displays a selection of tests available for the chosen variable(s), as shown in Figure 4. The full list of tests available are given in Table 1. The inference information can either be calculated using Normal theory or bootstrap methods (using the **boot** package, Canty and Ripley 2020).

2.4. Data wrangling

Researchers typically start a new analysis by creating a set of exploratory graphs, as described figure/inzight_inference.png

Figure 4: The INFERENCE window provides a selection of hypothesis tests for the chosen variables. In this case, these are `Infantmortality` (a numeric variable) and `Region` (categorical with six levels), so **iNZight** provides an ANOVA test.

Table 1: iNZight hypothesis test options.

Variable 1		Variable 2			
		NULL	numeric	2 level cat	2+ level cat
numeric		t-test[1]	–	t-test[3]	ANOVA
categorical	2 levels	single proportion	t-test[3]	χ^2-test[4,5]	χ^2-test[4,5]
	2+ levels	χ^2-test[2]	ANOVA	χ^2-test[4]	χ^2-test[4]

[1] One-sample
[2] Equal proportions
[3] Two-sample
[4] Equal distributions
[5] Additionally includes epidemiological output such as odds and risk ratios.

figure/inzight_data_mod.png

Figure 5: Here, the user is separating a column to create two new variable, with the preview displayed in the bottom-right. The relevent column names are underlined in red. The preview uses the first few rows of the data, and updates in real-time, reacting to changes the user makes, allowing them to experiment easily.

in Section 2.2. However, it is often not possible to get the correct graphs from the raw data, which might not be in the correct format. Applying transformations and other modifications to the data allows researchers to begin to explore the data, or approach it from a different perspective. **iNZight** contains two *data manipulation* menus: DATA and VARIABLES. The former acts on the data set as a whole, while the later creates modified versions of existing columns (variables).

In their book *R for Data Science*, Wickham and Grolemund (2017) describe many data manipulation methods including filtering, aggregation, and reshaping. They provide the **tidyverse** (Wickham *et al.* 2019) code for these actions, which **iNZight** uses behind-the-scenes to implement the behaviours. However, **iNZight** provides a GUI interface to these (often complex) methods, allowing users to quickly and easily filter by value, convert from *wide* to *long* form, or merge two related datasets together. In most cases, the interface allows users to fill out the fields which change according to previous selections, and at the bottom is a preview of what the data will look like, as demonstrated in Figure 5.

Supplementary to the dataset operations, the VARIABLE menu provides a selection of variable transformation and modification actions. For example, numeric variables can be converted to categorical (a common example is `Year`), or categoric variable levels can be renamed, reordered, and combined. Users also have the option of creating custom variables using R code, as well as renaming and deleting entire variables. In most cases, **iNZight** creates a *new* variable, for example converting `Year` to categorical yields the variable `Year.cat`, which makes the experience more exploration-friendly.

2.5. Special data types

Many data sets that beginners are exposed to are in 'tidy' format (Wickham and Grolemund 2017, chapter 12), such that rows are individual records and columns observations. However, there are some unique data types that are common for beginners to encounter, or form a core component of statistical anlysis. These data sets require unique graphics or other special handling to explore correctly, a task which **iNZight** has been extended to perform. Some examples are described here.

Complex survey designs

One of the more important data types for official statistics and population researchers are complex surveys, which require information about the survey's structure to provide valid graphs, summaries, and inferences. **iNZight** handles survey designs behind the scenes, requiring the user to specify the structure either manually (Figure 6) or by importing a special *survey design* file which can be distributed with the data. Once specified, the user can forget about the survey design and use **iNZight** as normal: survey weights are incorporated correctly into graphs, summaries, and data manipulation functions using the **survey** (Lumley 2004) and **srvyr** (Freedman Ellis and Schneider 2020) packages behind the scenes; the latter is a wrapper package providing **dplyr**-like syntax for surveys.

iNZight handles stratified, one- and two-stage cluster surveys. Instead of exposing the strata and cluster information in the dataset, some surveys provide this information using *replicate weights*, allowing variance estimation without exposing this information (Lumley 2010). **iNZight** provides an interface for specifying replicate weight designs if that is the case. In figure/inzight_survey_spec.png

Figure 6: Users can specify survey design information manually by filling in the fields. These will then be used throughout the session.

addition to these, **iNZight** also provides functionality to calibrate surveys with data from other sources to reduce the estimated variances (**?**).[4] Once again, this is performed once by the user and the used continually throughout the rest of **iNZight**.

The types of graphs available differ for survey data. For a single numeric variable, a *histogram* is displayed by default instead of a dotplot. For two numeric variables, a *bubble plot* is used, which is effectively a scatter plot with points sized by the respective weights of observations; alternatively, this can be displayed as a *hex* plot. Bar plots remain the same. Summaries accessed from the Get Summary button display the same information as standard data, but provide summaries of the population values along with the standard errors of the estimates, as shown in Figure 7.

Time series

Another important data type is *time series* in which the variable of interest is observed changing over time. Time information can be specified to **iNZight** either in a specially formatted column in the data, or manually by the user within the module itself. Then one or more variables can be displayed on the y axis versus time, typically connected by dots. Currently

[4] Find citation

`figure/inzight_survey_summary.png`

Figure 7: The SUMMARY window window provides simple summary statistics and, in the case of survey data, standard errors of these population estimates.

iNZight only supports time series with equally spaced and non-missing values.

iNZight's time series module provides capabilities for users to graph one or more time series on a graph, and automatically overlays a smoother (controlled by a slider) on each one. Additionally, the series can be decomposed to show the trend, seasonal, and residual components (using seasonal-trend decomposition using LOESS (STL)). Animations are available to help with understanding how the various components combine to form the final series, and a Holt-Winters' forecast can be obtained by choosing the *forecast* plot type (Holt 2004; Winters 1960).

Figure 8 shows the time series module with quarterly visitor arrivals data for several countries loaded. The software automatically detects the `Time` column ("Date") when loaded, and draws the displayed graph without any user interaction. Users have a choice between *additive* and *multiplicative* models, and a slider to control the 'smoothness' of the LOESS smoother (in red). From the SELECT VARIABLE list, one or more variables may be chosen, while the graph type is selected from the list on the right.

figure/inzight_time_series.png

Figure 8: The time series module.

Maps

Geographical data is commonplace, and is often of interest to researchers but it can be difficult to create the appropriate graphics. **iNZight** includes a *Maps* module for exploring two types of geographical data: point-based data, in which observations are associated with latitude and longitude locations (for example earthquakes); and regional maps for exploring data for specific regions which may be countries or areas within countries (regions, states, and so on). The functionality within the maps module call wrapper functions from the **iNZightMaps** package (Barnett and Elliott 2020).

For point-based observations, the points are overlayed on a map obtained using the **ggmap** package (Kahle and Wickham 2013). The maps module in **iNZight** lets users explore other variables in the dataset using size, colour, and transparency, as well as through faceting, again using the same interface controls as the base program. A demonstration of this modlue using New Zealand earthquake data is shown in Figure 9, where points are coloured by depth, sized by magnitude, and faceted by whether or not they were felt.

figure/inzight_quakes.png

Figure 9: The maps module showing New Zealand earthquakes sized by magnitude, coloured by depth, and subset by whether or not they were felt.

Regional data has the added complexity of *shape files* which describe the boundaries of areas. An example of this is country data, which requires shape files for countries in the dataset describing the world map. **iNZight**'s maps module lets users choose the type of map they need for the data, and proceeds to match labels between the two datasets using several matching techniques since: countries may be coded using full names or a code ("New Zealand" versus "NZ" versus "NZL"). Once initial set-up is complete, users are free to pick variables to graph, and regions of the maps are coloured appropriately. Alternatively, if longitudinal data is provided, *spark lines* can be drawn showing how the value of a variable changes over time in various regions.

Other data types and features

Besides these few examples, **iNZight** supports several other unique data types. Multiple response data arrises from "Choose all that apply" type survey questions, and need their own method of graphics to explore adequately. There is also a multivariate data add-on module for performing principle components analysis and generating the appropriate graphs. The model fitting module allows users to fit complex linear and generalised linear regression models to data, including that from complex survey designs. The regression model output is provided reactively as users add and remove variables, and a range of residual plots are available to explore.

Besides those examples listed above, **iNZight** has an add-on system (Section 4) that allows developers to extend the interface to suit various type of data or to perform specific analyses. Individual package developers, or research groups, can create custom modules that are then shared either publically or privately with others.

2.6. Code writing for getting started with R

One feature prominent in the other R GUIs is the coding interface, which differs significantly from **iNZight**'s. R Commander provides a prominent "script" box that users can enter custom code into, or is populated by menu-driven command boxes. Below the script box is an output terminal. Meanwhile, **Deducer** is added onto the existing R Console, and similarly provides menu-driven commands to run code in the console. Each of these GUIs require some familiarity with R coding and an understanding of simple statistical terms and methods. **iNZight**, however, is completely separate from the R console, providing an interface-only experience for beginners and users not interested in the code. All of the code used by various actions is stored and available for users to review their session and run it—with changes—in R manually.

The R script contains a history of all actions from importing the data, through transformations and manipulations, and any plots and summaries the user chose to save. The goal is to provide a record of what the user did, as well as something they can save to run in R themselves, editing where desired. This allows R learners to explore a dataset with a GUI tool and make the unique actions necessary, while researchers can generate a script to form the basis of a reproducible workflow.

Another feature is the inclusion of the R code box at the bottom of the interface (see Figure 1). This displays the code used to generate the current plot and, more importantly, can be edited by the user and run, somewhat similar to R Commander. When the user does so, the interface detects changes in the code and applies those changes to the GUI, providing a seamless way for users to begin experimenting with code whilst retaining the familiarity of the GUI. Users can also store the code for the current plot, which places it into the R script. A similar code box is displayed in the GET SUMMARY and GET INFERENCE windows, with plans to implement this behaviour throughtout **iNZight** in future.

The code used by **iNZight** uses a **tidyverse** (Wickham *et al.* 2019) workflow, as this provides an introduction to R with a simpler, verb-like syntax for data wrangling, and is used in the *R for Data Science* book (Wickham and Grolemund 2017). To demonstrate **iNZight**'s code-writing capabilities, Appendix A contains the script generated during the tour presented in this section.

3. Technical details

The interface for **iNZight** is developed entirely within R with the support of three main packages. **gWidgets2** (Verzani 2019) and **gWidgets2RGtk2** (Verzani 2020) provide a simple widget-based application programming interface (API) to building a cross-platform interface with R. The second package, **gWidgets2RGtk2**, is an interfacing package which provides a lower-level wrapper for the more complex functionality within the **RGtk2** package (Lawrence and Temple Lang 2010), which itself calls back to C libraries for the GTK+ windowing system (The GTK+ Team 2020). Together, these packages provide a platform- and library-independent API for creating a GUI with R.

The framework used to build **iNZight** uses object oriented programming (OOP), which is well suited to GUIs,[5]. **iNZight** uses *reference classes*, defined in the **methods** packages included with the base R distribution, and also used by **gWidgets2**, to describe individual components of the interface. Each 'panel' is a *class*, with individual buttons, methods (actions), and even smaller sub-components. OOP allows for *inheritance*, allowing developers to describe a general class which can be shared to several related components, but which may have

[5]Something about why?

figure/inzight_main_widgets.png

Figure 10: The reference class components of the **iNZight** interface, some of which are themselves made from several child objects.

different layouts or methods. Figure 10 shows the **iNZight** GUI with some of the major class components annotated.

In addition to these "visible" class components, others exist behind-the-scenes. The prominent one is the '`iNZDocument`' class which stores the state of the application, including the data set, variable selection, any survey design information, and plot settings. The '`iNZDataNameWidget`' component visible in the top-left of Figure 10 displays a list of documents the user can switch between and, from the DATA menu, merge several loaded datasets together.

The structure of each class is, in most cases, a set of attributes that the user can control, stored as *properties* of the class. There is also a set of *methods* which can be used by the class to react to user input or perform actions. Most components have a main *action*[6] method, which performs the primary function of the component. For example, the '`iNZFilterData`'

[6]Find a better word.

class contains a `filter_data()` method which takes the user's input and passes it to one of several *wrapper* functions, such as `iNZightTools::filterNumeric()`. A skeleton example of the Filter Data window class is shown in Listing 1. [7] In this oversimplified example, the user will be displayed a drop-down `gcombobox()` to choose a variable to filter on. When they click the Filter button, the data will be filtered and passed back to the main GUI. Here, the method uses a `switch()` function to select the appropriate wrapper function within the **iNZightTools** package based on the user's chosen value of "type". The actual class for the Filter Data method is much more complicated, and includes reactive components so only the relevant inputs are displayed to the user.

Each major component has a similiar structure to Listing 1, with calls to various functions, many of which come from other **iNZight*** packages. Plots are generated by calls to `iNZightPlots::inzplot()`, while data import is handled by `iNZightTools::smart_read()`. This import function uses the file extension to guess the file type and load the data using the appropriate methods. The wrappers enfore separation of the interface and data logic so that the GUI is only concerned with the input values.

A second advantage of having components calling external functions is that these wrapper functions can be designed to include the lower-level R code used to generate the result, which the GUI can fetch from the returned data and attach to the script discussed in Section 2.6. Here is an example of the result returned by `iNZightTools::smart_read()`:

```
R> library("iNZightTools")
R> data <- smart_read("nls.dta")
R> cat(code(data), sep = "\n")

haven::read_dta("nls.dta")
```

The `iNZightTools::code()` function returns the R code attached to the resulting object, allowing a user to see that the **haven** package (Wickham and Miller 2020) was used to read this Stata file (`.dta`). Beginner R users need only learn the one function—`smart_read()`—but can easily dive into the underlying code and edit it as necessary to access advanced options.

[7] Include a figure of this window, too?

```
iNZFilterData <- setRefClass(
    "iNZFilterData",
    propoerties = list(
        GUI = "ANY",
        data = "data.frame",
        type = "ANY",
        variable = "ANY",
        operator = "ANY",
        value = "ANY",
        ...
    ),
    methods = list(
        initialize = function(gui) {
            initFields(GUI = gui, data = gui$getActiveData())
            # ... construct GUI inputs ...
            # e.g.,
            type <<- gradio(c("Numeric value", "Factor levels", "Random"))
            variable <<- gcombobox(colnames(data))
            okbtn <- gbutton("Filter", handler = function(h, ...) filter_data())
        },
        filter_data = function() {
            filtered_data <- switch(svalue(type, index = TRUE),
                iNZightTools::filterNumeric(
                    data,
                    var = variable,
                    op = operator,
                    num = value),
                ...
            )
            GUI$update_data(filtered_data)
        }
    )
)
```

Listing 1: Reference class definition for filter window example.

Table 2: iNZight R package family

Package	Description
iNZight	The main package for the GUI
iNZightModules	An additional GUI package providing additional modules for the main **iNZight** program.
iNZightPlots	Provides plot function `inzplot()` along with `inzsummary()` for descriptive statistics and `inzinference()` for inference and hypothesis testing.
iNZightRegression	Plots and summaries of regression models, including from `lm()`, `glm()`, and `survey::svyglm()` objects.
iNZightTS	Time series visualisation, decomposition, and forecasting.
iNZightMR	Visualisation and estimation of multiple response data.
iNZightTools	A suite of helper functions for data process and variable manipulation.

While the GUI packages provide the structure of the visual GUI, it's the collection of R packages developed alongside **iNZight** that power the program. The main reason for creating separate packages was to force the separation of interface and data logic, but also to allow parallel development of a separate interface (Section 5.4) using the same wrapper functions. The collection of packages within the **iNZight** project are described in Table 2. Most of these packages have been designed with simple high-level interfaces that are both useful for connecting to the GUI, but also for beginners to use standalone.

3.1. Usage

At its core, **iNZight** is simply an R package that can be installed and run like any other, as covered in Section 5. Once installed, the main program can be started by creating a new instance of the main GUI class '`iNZGUI`', as demonstrated below.

```
R> library("iNZight")
R> ui <- iNZGUI$new()
R> ui$initializeGui()
```

For most users, however, the simpler wrapper function `iNZight()` can be called instead. This can optionally take a `data` argument, which will launch **iNZight** with the data loaded and ready to explore. Another use-case of the data argument could be to include within an R script used by a research group where the data needs to be loaded in a specific way (for example from a secure database). Users need only source a script similar to the following.

```
library("DBI")
con <- dbConnect(...)
tbl_data <- dbGetQuery(con, "SELECT ...")
library("iNZight")
iNZight(data = tbl_data)
```

For development purposes, the former startup method is recommended, as it provides access to the '`iNZGUI`' object created to explore states and trigger actions for easier testing. In these two commands, the first returns the dimensions of current data, while the second sets the first variable drop-down value to `height`.

```
R> dim(ui$getActiveData())

[1] 500  10

R> ui$ctrlWidget$V1box$set_value("height")
```

4. The add-on system

Most users will likely find all they need within the main **iNZight** interface, able to explore and visualise their data quickly and easily. However, there are cases where, as mentioned earlier, specific types of analyses and graphics are required for certain data types. Of course, the total range of data types available is close to limitless, and there are increasingly more R packages on CRAN every day providing new methods. Rather than requiring each new datatype or method to be manually coded into **iNZight** by the developers, we crafted an *Add on* system allowing anyone to create their own **iNZight** modules.

Installing existing add-ons is easy. Users can use the Module manager to add, update, and remove modules from our add-on repository, a custom URL, or a local file. In all cases, the file is placed in the `modules` directory, which users (and developers) can also place files manually. All files in this directory are then displayed in the Advanced menu of **iNZight**, and when opened have access to the current data and other aspects of the interface.

figure/inzight_demest.png

Figure 11: The prototype for a new Bayesian demographic modelling module for **iNZight**. In this example, life expectancy is estimated from death data.

The module files themselves describe a single class object which inherits from '`CustomModule`'. This parent class provides several methods, including the initialization of the module panel in the left-hand-side of the **iNZight** interface. Additional properties and methods can be written by the developers of individual modules. This opens up possibilities for teachers, research groups, or even R package developers themselves to write custom modules and distribute them to their desired audience.

As an example, Figure 11 shows a prototype for an upcoming Bayesian demographic modelling (Zhang *et al.* 2019) module which will be used by demographers to do small-area estimation, for example the estimation of death rates or life expectancy in small ethnic groups. This opens up advanced methods that were previously only available to proficient coders, who may now perform complex modelling procedures themselves. [8]

[8]Still need to rework this last paragraph.

5. Installation and availability

As an R package, **iNZight** may be installed manually from the R console like any other package. We have an R repository available at `https://r.docker.stat.auckland.ac.nz` which hosts the most up-to-date versions of our packages. Most of these are now on CRAN, and work continues to prepare the remainder. Since **iNZight** is a GUI, there are one or two additional system dependencies that need to be installed, with variations between operating systems, as discussed below.

5.1. Operating system-specific requirements

The GTK windowing system is a cross-platform project with libraries available on Windows, macOS, and Linux. However, the install process varies between operating system in both steps and complexity. On Windows, the necessary files are available in binary form, and can be installed *after* installing **iNZight**: the **RGtk2** package will prompt the user to download and install these binaries on the first run.

On macOS, users are required to install XQuartz and the GTK+ framework before manually compiling **RGtk2** themselves as, unfortunately, the binaries are no longer supported on CRAN. The complexity of this setup, and the lack of backwards compatibility of the macOS operating system, means we cannot officially support **iNZight** on macOS.

Finally, Linux comes in many flavours, each with different package managers and library names. However, the two dependencies are `xorg` and `gtk`, which are typically installed using the system package manager. For example, on Ubuntu 20.04, users can install the libraries thus:

```
$ apt-get install xorg-dev libgtk2.0-dev
```

Users of other operating systems should use the search functionality of their package manager to find the requisite libraries, or compile them themselves.

5.2. Windows installer

A large audience for **iNZight** is students new to statistics, who are unlikely to have the computer literacy[9] required by other GUIs to install and run the software (including R). To improve the accessibility of **iNZight**, we have built a custom installer that is effectively a self-extracting `.exe` file which includes a copy of R and the package library, so once installed **iNZight** is ready to go. This is by default installed into the user's `Documents\\iNZightVIT` directory.

In addition to the binaries and packages, the installer includes several shortcuts which can be double-clicked to launch R in a specific directory. This directory contains a `.Rprofile` file which automatically loads the **iNZight** package and launches the interface. It also hides the R console, so users are presented with just the GUI which is more familiar to them. When started from the script, R is passed a command to terminate the R session once the user has finished using **iNZight**.

The **iNZight** installer also includes an Update script to allow easy updating of the R packages. This allows novice users to update to the latest version without needing to use R or re-download the entire installer. Additionally we include an Uninstaller which removes **iNZight** from the user's system if they so desire.

5.3. Docker image

Docker is a development and deployment solution for developers to build, test, and share their projects (Merkel 2014). It allows developers to construct build chains with all dependencies included within a single image file which can be downloaded by users to run the program without installing a large set of dependencies. We have built a docker image for **iNZight**, allowing users of macOS and Linux to run the software without installing the system dependencies. The downside of this approach is that **iNZight** does not run as smoothly as it does natively, and also, as a GUI, requires a little more work from the user (particularly on macOS) to set up the necessary conditions for the app running in the container to access the host's graphical interface. More information can be found at `https://inzight.nz/docker`.

[9] another word, perhaps...

5.4. Online shiny version iNZight *Lite*

In recent years, many schools have adopted tablets or Chromebooks instead of Windows laptops, neither of which are capable of running R and, therefore, **iNZight**. To provide these students with equal opportunity, we created an online version of **iNZight** that uses **shiny** (Chang *et al.* 2021) as the GUI framework instead of GTK, named **iNZight** *Lite*.

Since most of the data-logic occurs in separate packages, porting **iNZight** to the web was simply a case of coding the interface elements and passing user inputs to the wrapper functions. This also means that the underlying code is the same between programs, so the *output* is the same in both cases, making it easier for teachers and researchers to use one or the other. We attempted to keep the interfaces as similar as possible, but there are obvious differences in the capabilities of the GUI toolkits.

The online version runs inside its own docker container on a remote Amazon Web Services server. Interested users could run the container locally by installing docker. Most users, however, will simply access the web interface by heading to `https://lite.docker.stat.auckland.ac.nz` in a browser, including on a tablet. There are also a set of URL parameters which can be passed to the **iNZight** *Lite* instance, including a URL for a dataset to automatically load.

Within the container, **shiny** (Chang *et al.* 2021) is used to create the visual controls and perform reactivity events. A user's data is stored on the server temporarily, and is only accessible from that user's session: it cannot be shared or accessed by other users. However, we still would not recommend users upload confidential or otherwise sensitive data; this would be better explored using either the desktop version or by running **iNZight** *Lite* locally. Research groups could host their own secure port of *Lite* with access to private data.

6. Summary and future work

Newcomers to statistics often need to learn both how to code using R whilst simultaneously learning the basic skills for data exploration, while many researchers need quick, easy methods to develop their projects. By providing an easy-to-use GUI, **iNZight** lets users focus on exploring and analysing data. Beginners can develop analysis skills before embarking on the more challenging part of learning to code. The software is *variable first*, meaning users do not need to first know or remember complicated statistical terminology to get the most from their data: the software provides a list of applicable methods given their current variable choice.

Similarly, data manipulation techniques such as filtering, renaming levels of factors, and even specifying survey designs, is all presented in simple step-by-step windows, many of which provide previews that help users to tweak the inputs to get the desired output. Many users may want to learn to code with R, and **iNZight** includes some simple tools for helping that migration: code writing to a session script, and a reactive code panel for modifying and running code for the current plot.

Statistics and data science is an ever expanding field, with new R packages added to CRAN daily. **iNZight** has an add-on system that developers outside of the development team can use to create and share modules for users to install and use on top of **iNZight**'s existing feature set. Since **iNZight** is available as a standalone program on Windows, package developers have an opportunity to engage previously unreachable audiences.

6.1. Future Work

Many new features and functionality are planned for the future of **iNZight**, the foremost being the ability to interact with more complex datasets, particularly those saved within a database, with as much processing done within the database as possible to speed up the interface for possibly large datasets. This, along with other advances, will make **iNZight** a useful tool for not only learners but researchers and organisations alike, including capabilities for the software to connect to secure databases behind a firewall and allowing researchers without coding skills access to it.

The main issue with **iNZight** at present is its reliance on GTK, which has been discontinued on macOS. Exploration into possible alternative frameworks is ongoing, with desire to develop a fully cross-platform application so users from all backgrounds can make use of the software.

Acknowledgments

iNZight is a free to use, open source software. The work would not have been possible without the support of: The University of Auckland; Census at School; Ministry of Business, Innovation, and Employment; Te Rourou Tātaritanga; Statistics New Zealand; and the Australian Bureau of Statistics. We also thank the technical support of the University of Auckland IT services for providing hosting services for our repository and Lite servers.

References

Barnett D, Elliott T (2020). ***iNZightMaps**: Map Functionality for **iNZight***. R package version 2.3.0.

Canty A, Ripley BD (2020). ***boot**: Bootstrap R (S-Plus) Functions.* R package version 1.3-25.

Chang W, Cheng J, Allaire J, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B (2021). ***shiny**: Web Application Framework for R.* R package version 1.6.0, URL `https://CRAN.R-project.org/package=shiny`.

Fellows I (2012). "**Deducer**: A Data Analysis GUI for R." *Journal of Statistical Software, Articles*, **49**(8), 1–15. ISSN 1548-7660. `doi:10.18637/jss.v049.i08`. URL `https://www.jstatsoft.org/v049/i08`.

Fox J (2005). "The R Commander: A Basic Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL `https://www.jstatsoft.org/article/view/v014i09`.

Freedman Ellis G, Schneider B (2020). ***srvyr**: **dplyr**-Like Syntax for Summary Statistics of Survey Data.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=srvyr`.

Holt CC (2004). "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages." *International Journal of Forecasting*, **20**(1), 5–10. `doi:https://doi.org/10.1016/j.ijforecast.2003.09.015`.

Kahle D, Wickham H (2013). "ggmap: Spatial Visualization with ggplot2." *The R Journal*, **5**(1), 144–161. URL `https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf`.

Lawrence M, Temple Lang D (2010). "**RGtk2**: A Graphical User Interface Toolkit for R." *Journal of Statistical Software*, **37**(8), 1–52. URL `http://www.jstatsoft.org/v37/i08/`.

Lumley T (2004). "Analysis of Complex Survey Samples." *Journal of Statistical Software*, **9**(1), 1–19. R package verson 2.2.

Lumley T (2010). *Complex Surveys: A Guide to Analysis Using R: A Guide to Analysis Using R*. John Wiley and Sons.

Merkel D (2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment." *Linux journal*, **2014**(239), 2.

Microsoft Corporation (2018). *Microsoft Excel*. `https://office.microsoft.com/excel`.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.

The GTK+ Team (2020). *GTK*. `https://www.gtk.org/`.

Verzani J (2019). ***gWidgets2**: Rewrite of **gWidgets** API for Simplified GUI Construction*. R package version 1.0-8, URL `https://CRAN.R-project.org/package=gWidgets2`.

Verzani J (2020). ***gWidgets2RGtk2**: Implementation of **gWidgets2** for the **RGtk2** Package*. R package version 1.0-7.1, URL `https://github.com/jverzani/gWidgets2RGtk2`.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the **tidyverse**." *Journal of Open Source Software*, **4**(43), 1686. `doi:10.21105/joss.01686`.

Wickham H, Grolemund G (2017). *R for Data Science*. O'Reilly Media. ISBN 978-1491910399. URL https://r4ds.had.co.nz/.

Wickham H, Miller E (2020). ***haven**: Import and Export SPSS, Stata and SAS Files*. R package version 2.3.1, URL https://CRAN.R-project.org/package=haven.

Winters PR (1960). "Forecasting Sales by Exponentially Weighted Moving Averages." *Management Science*, **6**(3), 324–342. doi:10.1287/mnsc.6.3.324.

Zhang J, Bryant J, Nissen K (2019). "Bayesian Small Area Demography." *Survey Methodology*, **45**(1).

A. Code history

The R code history generated during the demonstration in Section 2 is copied here.

```
# iNZight Code History

## This script was automatically generated by iNZight v4.0.3.9000
## -------------------------------------------------------------------------- ##

## This script assumes you have various iNZight packages installed.
## Uncomment the following lines if you don't:

# install.packages(c('iNZightPlots',
#                    'magrittr',
#                    'readr',
#                    'dplyr',
#                    'tidyr',
#                    'survey'),
#     repos = c('https://r.docker.stat.auckland.ac.nz',
#               'https://cran.rstudio.com'))

## -------------------------------------------------------------------------- ##

library(magrittr)  # enables the pipe (%>%) operator
library(iNZightPlots)

Gapminder <-
    readr::read_csv("C:\Users\Tom\Downloads\Gapminder.csv",
        comment = "#",
        col_types = readr::cols(
```

```
        BodyMassIndex_M = "c",
        BodyMassIndex_F = "c",
        Cellphones = "c",
        Femalesaged25to54labourforceparticipationrate = "c",
        Forestarea = "c",
        Governmenthealthspendingperpersontotal = "c",
        Hightotechnologyexports = "c",
        Hourlycompensation = "c",
        Incomeshareofpoorest10pct = "c",
        Incomeshareofrichest10pct = "c",
        Internetusers = "c",
        Literacyrateadulttotal = "c",
        Literacyrateyouthtotal = "c",
        Longtermunemploymentrate = "c",
        Poverty = "c",
        Ratioofgirlstoboysinprimaryandsecondaryeducation = "c",
        Renewablewater = "c",
        Taxrevenue = "c",
        TotalhealthspendingperpersonUS = "c"
    ),
    locale = readr::locale(
        encoding = "UTF-8",
        decimal_mark = ".",
        grouping_mark = ""
    )
) %>%
dplyr::mutate_at(
    c(
        "Country",
```

```
        "Region-Geo",
        "Continent",
        "Region",
        "Year_cat"
    ),
    as.factor
) %>%
dplyr::mutate_at(
    c(
        "BodyMassIndex_M",
        "BodyMassIndex_F",
        "Cellphones",
        "Femalesaged25to54labourforceparticipationrate",
        "Forestarea",
        "Governmenthealthspendingperpersontotal",
        "Hightotechnologyexports",
        "Hourlycompensation",
        "Incomeshareofpoorest10pct",
        "Incomeshareofrichest10pct",
        "Internetusers",
        "Literacyrateadulttotal",
        "Literacyrateyouthtotal",
        "Longtermunemploymentrate",
        "Poverty",
        "Ratioofgirlstoboysinprimaryandsecondaryeducation",
        "Renewablewater",
        "Taxrevenue",
        "TotalhealthspendingperpersonUS"
    ),
```

```
        as.numeric
    ) %>%
    dplyr::rename(Region.Geo = "Region-Geo")

Gapminder.filtered <-
    Gapminder %>%
    dplyr::filter(Year_cat %in% c(
        "[1960]",
        "[1964]",
        "[1968]",
        "[1972]",
        "[1976]",
        "[1980]",
        "[1984]",
        "[1988]",
        "[1992]",
        "[1996]",
        "[2000]",
        "[2004]",
        "[2008]"
    )) %>%
    droplevels()

inzplot(Infantmortality ~ GDPpercapita | Year_cat,
    colby = Region,
    sizeby = Populationtotal,
    data = Gapminder.filtered,
    xlab = "GDP per Capita (log scale)",
    ylab = "Infant Mortality",
```

```
    col.fun = "contrast",
    alpha = 0.4,
    transform = list(x = "log10"),
    main = "Infant Mortality Over Time"
)

inzinference(Infantmortality ~ Region | Year_cat,
    g1.level = "[2004]",
    data = Gapminder.filtered,
    hypothesis.test = "anova"
)

Gapminder.filtered.separated <-
    data %>% tidyr::separate(
        col = "Region.Geo",
        into = c(
            "main_region",
            "part_region"
        ),
        sep = " - ",
        extra = "merge"
    )

## Load example data set
data(apiclus2, package = 'survey')

## -------------------------------------------------------------------------- ##
## Exploring the 'apiclus1_ex' dataset
```

```
apiclus1_ex <- apiclus1


## create survey design object
apiclus1_ex.svy <- survey::svydesign(ids = ~dnum, fpc = ~fpc, nest = FALSE,
    weights = ~pw, data = apiclus1_ex)


inzsummary(~stype,
    design = apiclus1_ex.svy
)


## Load example data set
data(visitorsQ, package = 'iNZightTS')


## -------------------------------------------------------------------------- ##
## Exploring the 'visitorsQ_ex' dataset


visitorsQ_ex <- visitorsQ
```

Affiliation:

Tom Elliott

School of Health

Victoria University of Wellington

Wellington, New Zealand

and

Department of Statistics (Honorary)

University of Auckland

Auckland, New Zealand

E-mail: `tom.elliott@auckland.ac.nz`

URL: `https://people.wgtn.ac.nz/tom.elliott`

Journal of Statistical Software `http://www.jstatsoft.org/`

published by the Foundation for Open Access Statistics `http://www.foastat.org/`

MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd

`doi:10.18637/jss.v000.i00` *Accepted:* yyyy-mm-dd