Journal of Statistical Software

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

iNZight: A Graphical User Interface for Democratising Data Visualisation, Exploration, and Analysis

Tom Elliott
Victoria University of Wellington

Chris Wild
University of Auckland

Daniel Barnett
University of Auckland

Andrew Sporle
iNZight Analytics

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. R GUIs are available, but they require some degree of experience with R and a familiarity with statistical concepts. **iNZight** is a GUI-based tool written in R that enables both students and researchers 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 for common tasks. **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, democratisation.

1. Introduction

The open source statistical programming environment R (R Core Team 2020),used throughout statistics and data science, is supported by a repository of thousands of free packages providing access to the latest statistical techniques and graphics, plus a wide range of other tasks. Among these packages are several graphical user interfaces (GUIs) providing point-and-click methods for interacting with R to create graphs, test hypotheses, and access 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 user-editable R code, while **Deducer** extends the R console with menus from which users can access GUI interfaces for a range of methods. These two tools let users perform known procedures using point-and-click interfaces without the need to remember function and argument names. They do, however, require that users not only install R along with the dedicated packages, but also have a general understanding of the underlying methods and how to use them. This excludes users new to statistics who do not yet know the necessary terminology to benefit from the GUI.

An alternative design is to let users choose the variables, and have the software present a selection of applicable methods, using a good default where possible. This approach is used by **iNZight**, a GUI built with R that allows non-coders to explore, visualise, and analyse data. Designed with "variable-first" concepts, **iNZight** is more approachable and has a focus on exploration, with a special emphasis on graphics. The software encourages users to explore the variables in their dataset using automated plots, with summary statistics and inference information accessed via designated buttons. These summaries and inferences display information relevant to the chosen variables, so users do not need to know in advanced that the test used to compare the means of two groups is a t-test: the software provides a list of applicable tests. This concept is used throughout **iNZight**, making it ideal for not only beginners new to statistics, but to researchers who may need to perform simple tasks on an infrequent basis as well. Additionally, it makes performing some data analysis tasks more accessible to organisations who would otherwise need to train individuals within their organisation, or hire specialists—both of which require time and money. Since **iNZight** is more approachable and easier to use, it makes several data analysis tasts more accessible, so organisations can use **iNZight** as an in-house research tool rather than training researchers to use more complex systems or contracting out these tasks, both of which cost time, money, or both.

Like other R GUIs, there is a code component to **iNZight**. R Commander includes a dedicated code box, while **Deducer** runs on top of the R console using the existing environment. With **iNZight**, however, code is evaluated "behind the scenes", and is not directly editable by users. Every 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, allowing researchers to quickly develop a scripted, repeodicible workflow for working with data that can easily be modified as needed. The script can also be used as a stepping stone for learning to code in R, particularly useful for the large part of **iNZight**'s audience: statistics students.

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 organisation, including the Australian Bureau of Statistics, …**ANDREW TO FILL**. 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 built as a tool for high school statistics students, and later incorporated into introductory statistics courses in New Zealand, however it has since matured and is now useful as a rapid research development tool for community and government research groups. The interface is minimal and provides instant feedback from user actions, making it intuitive to use so users can easily start exploring their data. The *variable-fist* paradygm makes it easy to learn new statistical concepts—such as hypothesis testing—by focusing on the output first,

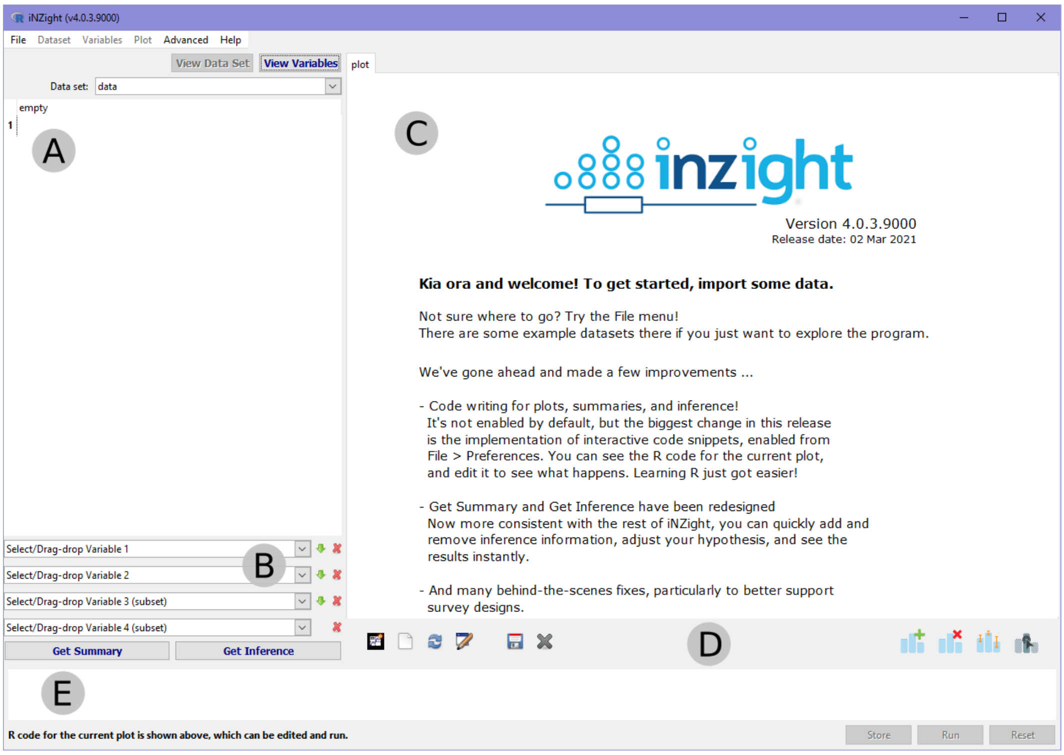

Figure 1: The **iNZight** GUI landing page presents users with a few controls. The labeled areas are: (A) the active data set is displayed prominently; (B) variable control boxes provide users either drag-and-drop from (A), or select from dropdowns; (C) graphs are displayed in the graphics window; (D) 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.

and consequently makes it very easy to pick up after a period of non-use, typical of critical but seldomly performed tasks within organisations. Ease-of-use is achieved through familiar controls such as *drag-and-drop*, *drop-down* selection, and *slider bars*. **iNZight**'s variable-first approach lets users choose the variables they are interested in and the software decides on the best action to take (or provides a small set of choices). The simplest way to explore **iNZight**'s features is by demonstration.

2.1. Loading data

Datasets come in a wide range of formats, some of which are traditionally software-dependent (for example Excel, Microsoft Corporation 2018, stores files in Excel (.xlsx) format). Fortuately, there are 1000's of R packages on the Comprehensive R Archive Network (CRAN), amongst which are some dedicated to importing most of the common (and indeed many

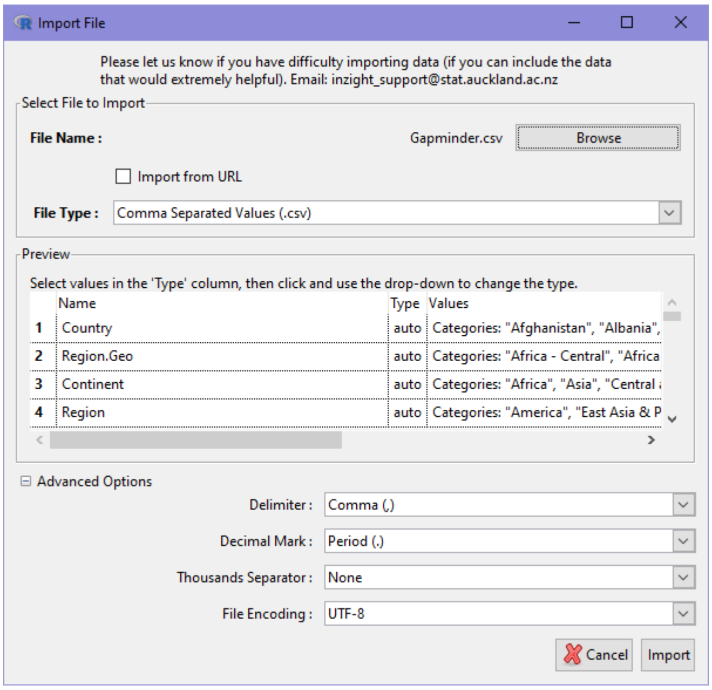

Figure 2: Load Data window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

uncommon) file formats. Usually, R users would need to first recognise or look up the file extension, find an appropriate package, then decipher the documentation to import an unusual file. **iNZight** provides a simple Load Data window from which users only need select the file to import: the software detects the file type from the file extension and reads the file if the format is supported. Currently, **iNZight** supports files in comma separated values (CSV), tab-delimited, Excel, SAS, Stata, SPSS, R-data, and JSON formats. If the file is readable by **iNZight**, a preview is displayed for the user to check before proceeding with the import, as demonstrated in Figure 2 which shows the Load Data window for a CSV file.

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 can use trial-and-error if they are not sure what the necessary inputs are. This is particularly useful for encoding, which is difficult to find out manually.

Lastly, the data preview allows users to override the default variable types. This is particularly useful when importing a dataset with coded factors (e.g., values $1, 2, 3$ instead of "A", "B", "C"). At the time of writing, **iNZight** handles numeric, categorical, and date-time formats—with more on the way.

2.2. Creating graphs

Graphics provide the core of **iNZight**'s user experience; indeed, the very first prototype of **iNZight** was simply a drag-and-drop of variables onto slots to create a graph—everything else came later. Behind the scenes, **iNZight** uses variable types (numeric, categorical, or date-time) to determine the appropriate graph. For example, a single numeric variable are visualised with a dot plot, while a categorical variable uses a bar chart. While this may seem obvious to experienced statisticians, beginners cannot be expected to immediately grasp the concept of a "numeric" variable, and why a dot plot is used instead of a bar chart. By removing this step, teaching and learning can focus on questions such as *"why does this variable produce this graph?"*. It also means experienced users can very quickly look at range of graphs without needing to account for different data types.

The control panel (**B** in Figure 1) lets users choose up to four variables: the first of these (*Variable 1*) specifies the *Primary Variable of Interest* (or *Outcome Variable*). The remaining three variable boxes are for exploring what variables influence the first. For example, 'height' might be the primary variable, so selecting it will produce a *dot plot* of height. If a second variable chosen is categorical, for example 'ethnicity', then we are shown a dot plot of height for each ethnicity, stacked vertically. If *Variable 2* is *numeric*, such as 'age', we see a graph of height versus age: *Variable 1* (height) becomes the y-variable in the scatter plot. The last two variable slots are subset variables that quickly and easily facet the plot, allowing users to explore more complex relationships and interactions. Any numeric variables used for either of the subsetting variables are automatically cut into four class intervals.

For a deeper exploration of variable relationships, there is an entire panel dedicated to plot modifications: ADD TO PLOT. This is accessed either from the PLOT menu, or from the

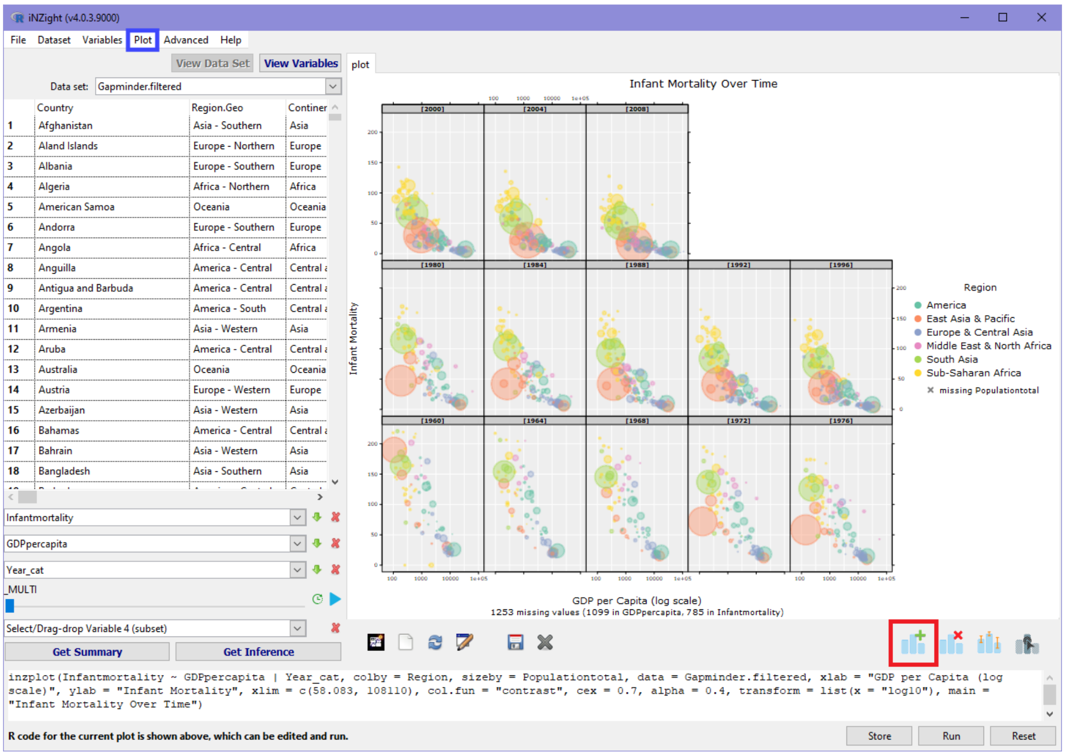

Figure 3: Demonstration of plot modifcations available from **iNZight**'s Add to Plot menu. The Add to Plot button, highlighted in red, opens a panel giving user control over colours, size, shape, labels, and much more. This can also be accessed from the plot menu, boxed in blue.

button in the Plot Toolbar (boxed in red in Figure 3). Users can choose from a selection of alternative plot typesbased on the variable(s) selected, as well as choose a colour variable, sizing variable, plot symbols, trend lines, change axis labels and limits, and much more. The possible choices 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?". This way, beginners can learn about the software while they explore the data: they are not limited by a lack of knowledge or 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 (log) GDP, region (colour), population (point size), and year (faceting).

2.3. Summaries and inference

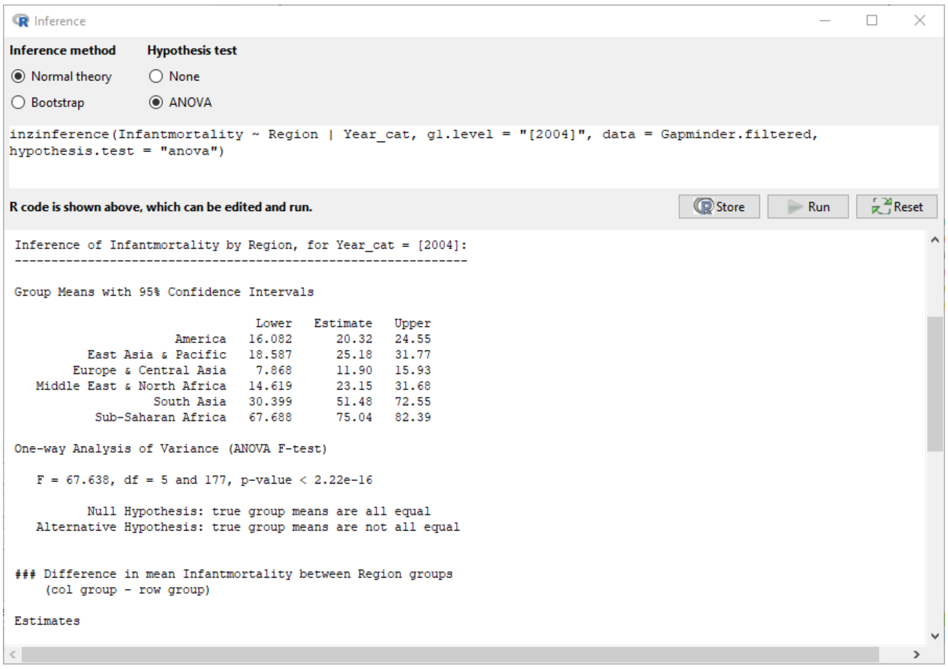

Figure 4: The INFERENCE window provides a selection of hypothesis tests for the chosen variables. In this case, these are `Infantmortality` (a numeric variable) and `Region` (categorical with six levels), so **iNZight** provides an ANOVA test.

To supplement the graphical display, **iNZight** provides two textual output modes: *summary* and *inference*, accessed from the GET SUMMARY and GET INFERENCE buttons, respectively, below the control panel. Summary information includes basic statistics about variable(s) in the graph. Dot plots are summarised by means, standard deviations, and quantiles. Bar chart summaries display a table of counts and percentages. Scatter plots provide the formula for any fitted trend lines, along with the correlation between the variables. If subset variables are present, then individual summaries for each subset is provided.

The inference information includes estimates, confidence intervals, and any applicable p-values for quantities such as means and proportions. For performing hypothesis testsing, **iNZight** displays a set of tests applicable to the chosen variable(s), as shown in Figure 4 (the full list of tests available are given in Table 1). Inference information is either calculated using Normal theory or Bootstrap methods (using the **boot** package, Canty and Ripley 2020), as chosen by the user, with work on-going to add Bayesian inference methods.

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.

2.4. Data wrangling

Researchers typically start a new analysis by creating a set of exploratory graphs, as described in Section 2.2. However, it is often not possible to get the desired graphs from the raw data, as it may not be in the correct format. Often, data transformations are required before any useful analysis can begin (for example converting numeric codes to categorical variables) or to explore from a different perspective. **iNZight** contains two *data manipulation* menus: Data and Variables, for manipulating the full dataset and individual columns (variables), respectively.

In their book *R for Data Science*, Wickham and Grolemund (2017) describe many data manipulation methods including *filtering*, *aggregation*, and *reshaping*. They demonstrate the **tidyverse** (Wickham *et al.* 2019) code for these actions, which **iNZight** uses behind-the-scenes to implement users' chosen actions. **iNZight** provides a GUI interface to these (often complex) methods, enabling users to quickly and easily filter by value, convert from *wide* to *long* form, or join two datasets together. In most cases, the interface evolves from top-to-bottom as the users first chooses, for example, the variable, with subsequent inputs tailored to previous choices, and so guiding users through the action. At the bottom of many data manipulation windows is a preview of the data after transformation, as demonstrated in Figure 5. A full list of available methods is provided in Appendix A.1.

The Variable menu gives users access to a range of variable transformation and modification actions. For example, numeric variables can be converted to categorical (a common example

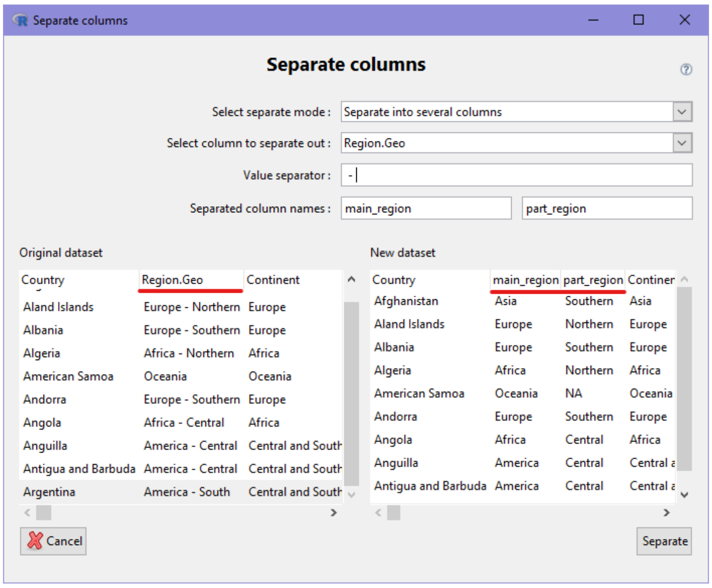

Figure 5: Here, the user is separating a column to create two new variable, with the preview displayed in the bottom-right. The relevent column names are underlined in red. The preview uses the first few rows of the data, and updates in real-time, reacting to changes the user makes, allowing them to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

is `Year`), or categorical variable levels can be renamed, reordered, or combined. Users can also creating new variables based on existing ones, as well as rename or deleting existing ones. **iNZight** creates a *new* variable for each action—converting `Year` to categorical yields the variable `Year_cat` for example—which makes the experience more exploration-friendly. Appendix A.2 gives a list of available variable manipulation methods.

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 data types that beginners may encounter, or form a core component of statistical anlysis. These data sets require special graphics and different handling to explore correctly, a task which **iNZight** has been extended to perform. Some examples are described here.

Complex survey designs

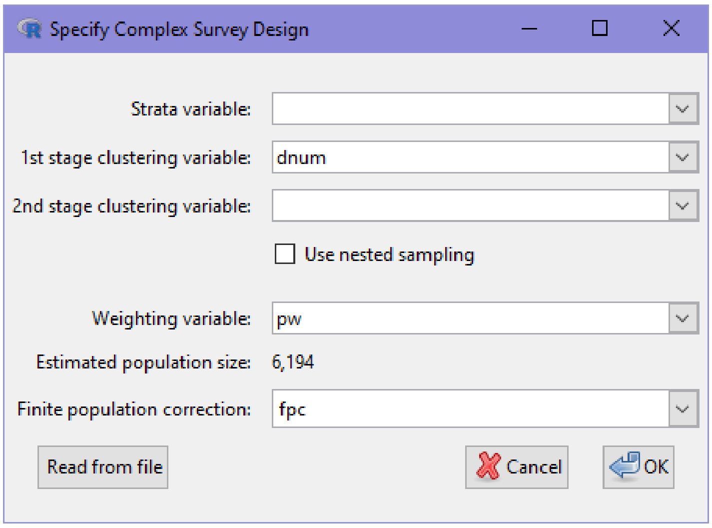

Figure 6: Users can specify survey design information manually by filling in the fields. These will then be used throughout the session.

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.

iNZight handles stratified, one- and two-stage cluster surveys, along with *replicate weight* designs, a format common in demographic surveys where clustering information is not included in the data, while still allowing variance estimation (Lumley 2010);[1]. **iNZight** provides an interface for specifying replicate weight designs in that case. **iNZight** can also calibrate surveys with data from other sources to reduce the estimated variances (**?**). Once again, this is performed once by the user and is 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 dot plot. For two numeric variables, a *bubble plot* is used,

[1] I don't like this sentence

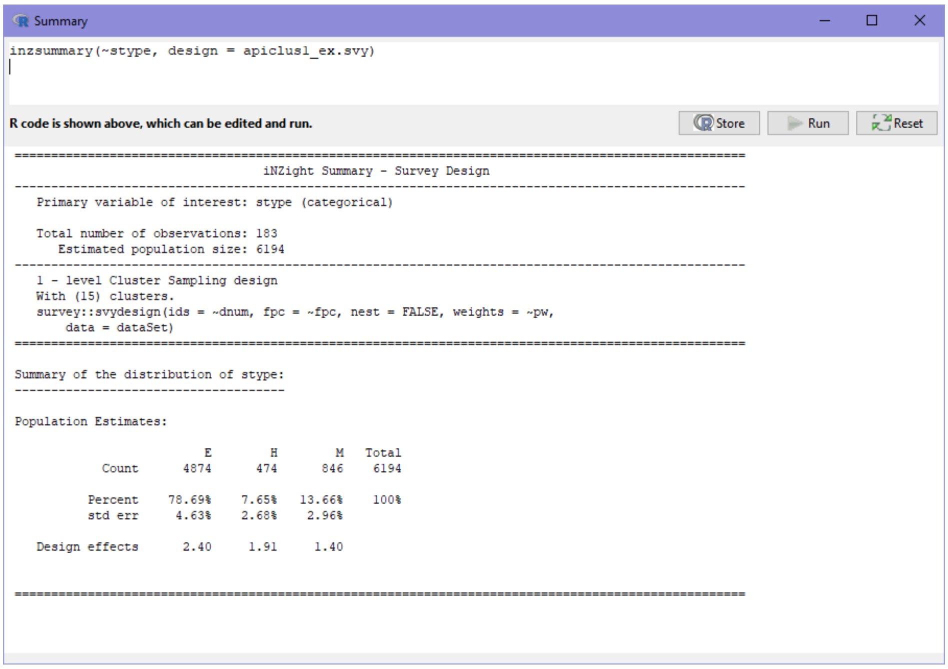

Figure 7: The SUMMARY window window provides simple summary statistics and, in the case of survey data, standard errors of these population estimates.

which is effectively a scatter plot with points sized by the weights of respective observations; alternatively, this can be displayed as a *hex-bin* plot, particularly when there are many observations. Bar plots are still used for surveys. Summaries display the same information as before (Section 2.3), but provide estimates and standard errors of the population values, as shown in Figure 7. Similarly, inferences and hypothesis tests are performed for the population, and thus include additional uncertainties from the survey design.

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** in its dedicated *Time Series* module, either in a specially formatted column in the data, or manually by the user within the module itself. Currently **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 to see how values change over time, and automatically overlays a smoother (con-

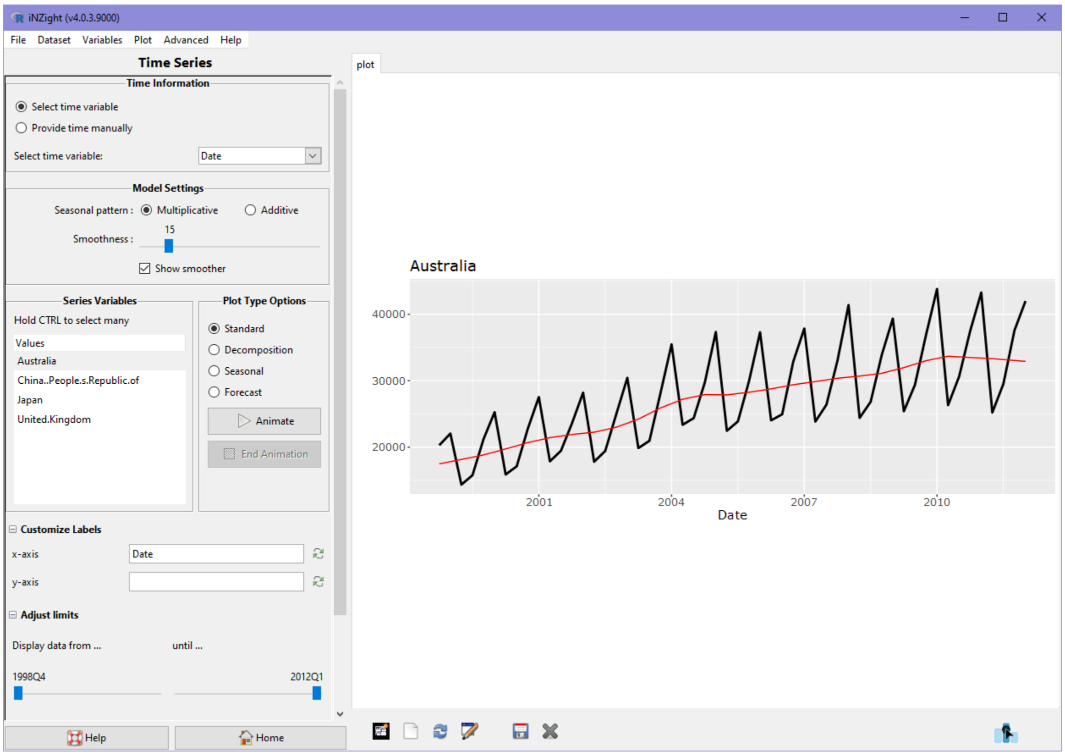

Figure 8: The time series module.

trolled by a slider) on each. 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. 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 ('Plot type Options').

Maps

Geographical data is particularly important for looking at regional effects, or the distribution of location-based events; however, it can be difficult to create appropriate graphs which often require sourcing maps or shape files. **iNZight** features 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 related to fixed regions which may be countries or areas within countries (regions, states, and so on). The functionality within the maps module calls 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 the same techniques used for scatter plots: size, colour, and transparency, and faceting, all using interface controls very similar to the base program. A demonstration of this module using New Zealand earthquake data is shown in Figure 9,[2] where points are coloured by depth, sized by magnitude, and faceted by whether or not they were felt.

Regional data has the added complexity of requiring *shape files* to describe the boundaries of areas. For example, New Zealand can be divided into regions (Auckland, Otago, etc), which have physical boundaries that can be described by an external shape file, allowing data relating to New Zealand's regions to be shown on a 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: 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 map 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 special data types. *Multiple response* data arises from "Choose all that apply" type survey questions, and need their own

[2] Sourced from `https://www.geonet.org.nz/`

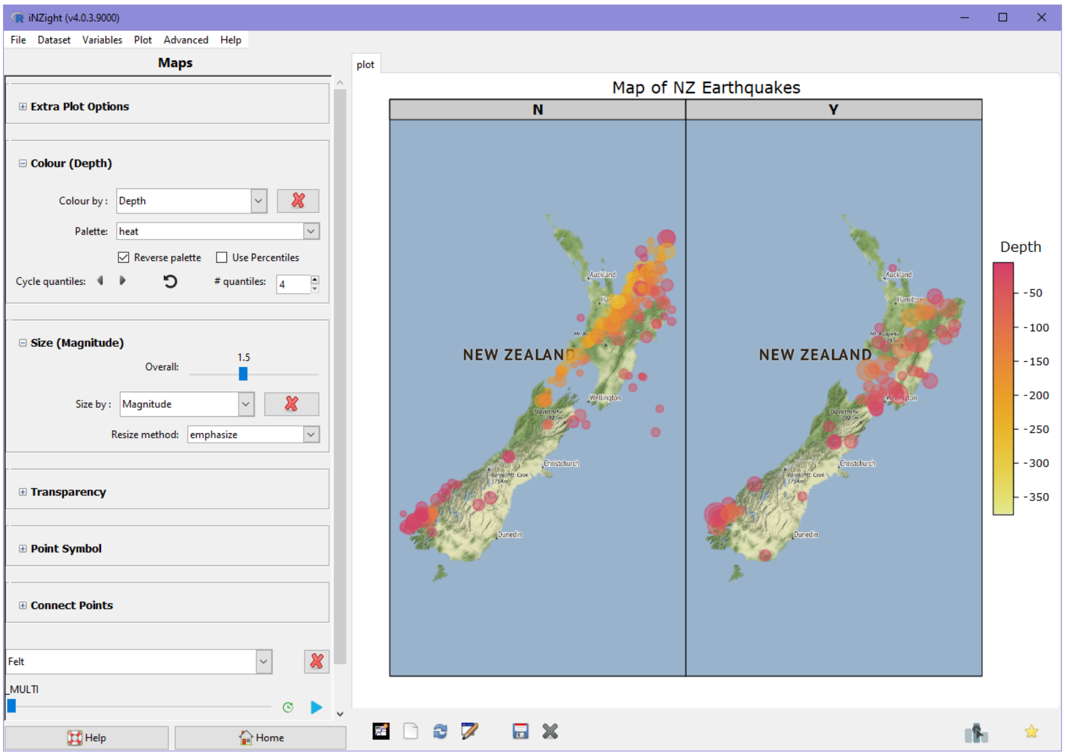

Figure 9: The maps module showing New Zealand earthquakes sized by magnitude, coloured by depth, and subset by whether or not they were felt.

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 (e.g., a principle components plot). The model fitting module allows users to fit complex linear and generalised linear regression models to data (including 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 and help users to quickly fit a model with any necessary transformations.

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

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 into which code appears when using the command boxes, or users can enter their own code, and below this is an output terminal. In contrast, **Deducer** is added onto an existing R Console, providing menu-driven commands to run code in the console. Each of these GUIs require familiarity with R coding and an understanding of simple statistical terms and methods in order to navigate the menus. **iNZight**, however, is completely separate from the R console, providing an interface-only experience for beginners and those users not interested in coding. Code generated by various actions behind the scenes is stored and available for users to review and run—with changes—in R manually.

The R script contains a history of all actions executed by the user, from importing the data, through transformations and manipulations, to 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 lets users explore a dataset with a GUI tool to quickly start an anlysis that can be used as the basis of a reproducible workflow.

A more advanced feature is 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, but remains otherwise limited in functionality. 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, adding it to the R script. A similar code box is displayed in the Get Summary and Get Inference windows, with plans to implement this behaviour throughout **iNZight** in future.

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 B contains the script generated during the tour presented in this section.

3. Technical details

iNZight's interface is written with R using three main supporting packages. **gWidgets2** (Verzani 2019) provides a simple widget-based application programming interface (API) to building a cross-platform interface with R, with wrappers for Tcltk, Qt, and GTK2.[3] We chose GTK2, as it was the most feature rich and—at the time—had the best cross-platform support (see Section 5.1). The GTK2 binaries are accessed through R using the **RGtk2** package (Lawrence and Temple Lang 2010), with commands translated from **gWidgets2** using the **gWidgest2RGtk2** package (Verzani 2020). Together, these packages provide a platform-independent API for creating a GUI with R.

The GUI for **iNZight** uses an object oriented programming (OOP) framework in R called *reference classes* (from the **methods** packages included with the base R distribution). The same framework is used by **gWidgets2** to describe individual components of the interface. Each piece of the GUI is a *class*, with individual buttons, methods (actions), and even smaller sub-components. OOP allows for *inheritance*, so developers can describe a general class which can be shared by several related 'child' components, but which may have different layouts or methods; this is what drives the add-on system (Section 4). Figure 10 shows the **iNZight** GUI with some of the major class components annotated.

In addition to the "visible" class components, others exist behind-the-scenes, the main one being 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, allowing the user to switch between them, from the DATA menu, several loaded datasets can be merged 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 method which performs the primary function of the component. For example, the '`iNZFilterData`' class contains a `filter_data()` method which takes the user's input and passes it to an appropriate *wrapper*

[3] citations? URLs? styling? check R Command + Deducer papers.

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

function. A skeleton example of the Filter Data window class is shown in Listing 1. In this oversimplified example, the user is given a drop-down `gcombobox()` to choose a variable to filter on. When they click the Filter button, the data is filtered and passed back to the main GUI. The method uses `switch()` 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 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()`. The wrappers enforce separation of the interface and data logic so that the GUI is only concerned with the input values.

Another advantage of having components calling external functions is that the wrapper functions can 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 described in Section 2.6 all while keeping the GUI and data-oriented code separate. Here is an example using `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.

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. However, it also

```
iNZFilterData <- setRefClass(
    "iNZFilterData",
    fields = 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.

allowed for the 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 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 calling the function of the same name:

```
R> library("iNZight")
R> iNZight()
```

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, it is preferable to initialize the GUI object manually:

```
ui <- iNZGUI$new()
ui$initializeGui()
```

This way users have access to the 'iNZGUI' object and can explore states and trigger actions for easier testing. In these next 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

For most users, the main **iNZight** program will have all they need to explore, visualise, and perform simple analyses on their data. Some, however, may require access to special analyses not built in to the base program. 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 that can connect to new or existing R packages available on CRAN or elsewhere.

Installing existing add-ons is easy. From the Module manager users can add, update, and remove modules from our add-on repository,[4] a custom URL, or a local file. In all cases, the file is downloaded to the `modules` directory; users can also place files there manually, if desired. All files in this directory are displayed in the Advanced menu of **iNZight**, and when opened have access to the **iNZight** interface, including the dataset imported by the user.

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

[4] https://github.com/iNZightVIT/addons

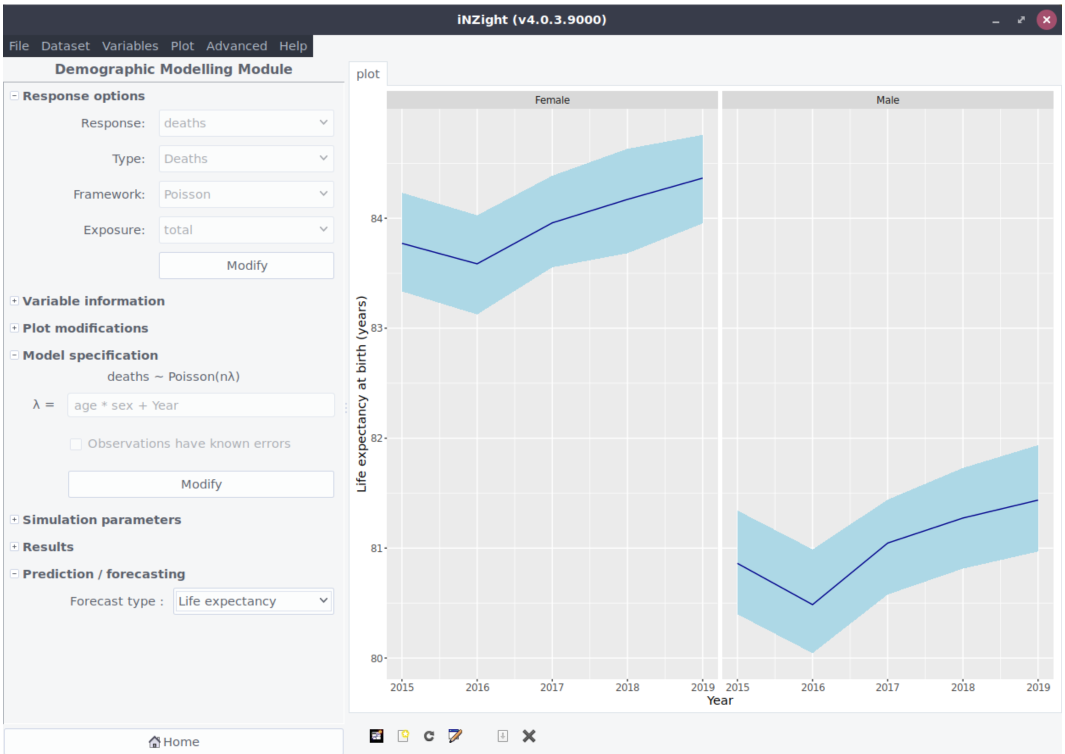

Figure 11: The prototype for a new Bayesian demographic modelling module for **iNZight**. In this example, life expectancy is estimated from death data.

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 to distribute to their audiences.

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. In the example, we have estimated life expectancies from death count data, which is traditionally a complicated . . . Full details for this module will be published once the full version is complete. **Pull some more details from dedicated report.**

5. Installation and availability

As an R package, **iNZight** may be installed manually from the R console like any other package. We maintain 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 and submit the remainder. Since **iNZight** is a GUI, there are additional system dependencies that need to be installed, which vary 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 systems 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 the first time the package is loaded.

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 main dependencies are `xorg` and `gtk2.0`, which are typically installed using the system package manager. For example, on Ubuntu 20.04, users can install the libraries using:

```
$ 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. There are several other system dependencies which need to be installed for some features of **iNZight**. For the latest list, check `inzight.nz/install`. Note that Linux users can install **iNZight** by running the Windows installer (Section 5.2) under Wine.[5].

5.2. Windows installer

[5] https://www.winehq.org/

A large audience for **iNZight** is students new to statistics, who are unlikely to have the computer skills required by other R GUIs to install and run the software (including R). To improve the accessibility of **iNZight**, we deploy an 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 a GUI, as they will be most familiar with. 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—since **iNZight** is standalone, this simply deletes the folder and any shortcuts.

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`.

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 developed 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 students and researchers to use one or the other. We attempted to keep the interfaces as similar as possible, but some differences are unavoidable due to the constraints of the individual 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, can access the web interface by heading to `https://lite.docker.stat.auckland.ac.nz` in a browser on a computer or tablet. There is a set of URL parameters which can be passed to the **iNZight** *Lite* instance, including a URL for a dataset to automatically load, so for examples it is possible to store datasets on a server with a list of URLs to launch Lite with the chosen data loaded. For example, `https://lite.docker.stat.auckland.ac.nz/?url=https://inzight.nz/testdata/nhanes.csv&land=visualize`. An example is this *data set listing* in action can be seen at `https://www.stat.auckland.ac.nz/~wild/data/Rdatasets/`.

Within the container, the **shiny** package 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. **Unhappy with this para either ...**

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 tools to get new projects started. By providing an easy-to-use GUI, **iNZight** lets users focus on exploring and analysing data, and beginners can develop interpretation skills before embarking on the more challenging part of learning to code. The software is *variable first*, so 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 selection(s), and effecitvely helps to guide them through the analysis.

Similarly, data manipulation techniques such as filtering, renaming levels of factors, and even specifying survey designs, are all presented in simple step-by-step windows, many of which provide previews, helping users to tweak the input controls to get the correct 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 **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 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 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; the Australian Bureau of Statistics; and iNZight Analytics. We also thank the technical support of the University of Auckland's Digital Solutions group 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/.

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. Data wrangling methods

iNZight features the following methods for transforming and manipulating data.

A.1. Dataset

Transform, mutate, etc.

A.2. Variables

Create, rename, change, etc.

B. 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@vuw.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