Journal of Statistical Software

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

iNZight: A Graphical User Interface for Visualisation and Exploration of Data with R

Tom Elliott
Victoria University of Wellington

Chris Wild
University of Auckland

Daniel Barnett
University of Auckland

Andrew Sporle
University of Auckland

Abstract

Getting started with data science is a daunting task, particularly when it requires a large amount of coding before you can even start looking at data. Graphical user interfaces (GUIs) have often been used as a way of proving novice users the ability to interact with complex systems without the need for coding. However, many of these themselves have steep learning curves to understand how to make the software do what's needed, and do not provide a pathway to more standard and flexible methods, such as coding. **iNZight** is a GUI based tool written in R that provides students of statistics and data science the opportunity to interact with data and explore without first learning to code. The tool is designed to be easy to use, with logical interactions and clever defaults. However, it also provides some more complex features to manipulate and analyse data, and further provides a code history of the actions performed, creating a pathway between GUI and learning to code for those interested in progressing into the more open and exciting world of data science.

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

1. Introduction

The R programming environment (R Core Team 2020) is used throughout statistics and data science due to it being open source, easy to learn, and backed by a huge package repository to solve even the most unique of problems. Included in these packages are several graphical user interfaces (GUIs) providing point-and-click methods for creating graphs, performing hypothesis tests, and a range of other actions, with two prominent examples being R Com-

mander (Fox 2005) and **Deducer** (Fellows 2012). These tools work by providing *additional functionality* to R, namely with menus allowing users to choose and use methods without remember the indivual function name and argument order. However, this requires a certain level of understanding of what these tests are for, excluding many beginners to statistics and data science.

An alternative approach is to work variable-first, such that users choose variables they are interested in, and then choose from an automatically curated list of options to perform. **iNZight** uses this approach, and presents users with an exploration-focussed interface, with the primary emphasis on graphics which is at the core of exploratative data analysis. Once basic visual exploration has taken place, obtaining simple summary information is as simple as clicking a button, as are inference statistics, including hypothesis testing, for which the user can choose from a small selection based on the chosen variables. This makes **iNZight** ideal for users who are completely new to statistics and data science, as well as those users who seldom perform these tasks and might otherwise struggle to remember the terminology required if using alternative software.

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, and using the environment as-is. With **iNZight**, however, code happens "behind the scenes", and is not directly editable by users. A code history is stored with every action the user makes, allowing those interested to see what the code looks like, and to begin bridging the gap between learning to explore data and learning to code. [[any references about learning to code using a GUI ?]]

Due in part to its ease of use, **iNZight** has been adopted throughout New Zealand's statistical education program. Final year high school students are introduced to basic statistical concepts using **iNZight**, including a foray into time series analysis. Universities accross the country have also begun to use **iNZight** in both introductory and some advanced statistics courses. This paper provides an overview of some of the main features of **iNZight**, along with technical details, an introduction to its *Add-on* system, and description of the install process.

2. A tour of iNZight's features

The primary audience for **iNZight** are students of statistics and data science, but secondary groups include small research groups with small budgets and organisations which perform analyses infrequently. The simplicity of the interface, shown in Figure 1, means its easy to remember how to use the program after a period of non-use, unlike more specialised software which can cost time to relearn. We have done this by making the interface as intuitive as possible, with few self-explanatory controls using basic mechanisms such as drag-and-drop or selection from a drop-down box. **iNZight** also uses a *variable first* approach, meaning users choose the variables they are interested in, and the software displays relevant actions. The easiest way to demonstrate **iNZight**'s features is by demonstration. In this section, we take a tour through the software from simple to complex.

2.1. Loading data

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

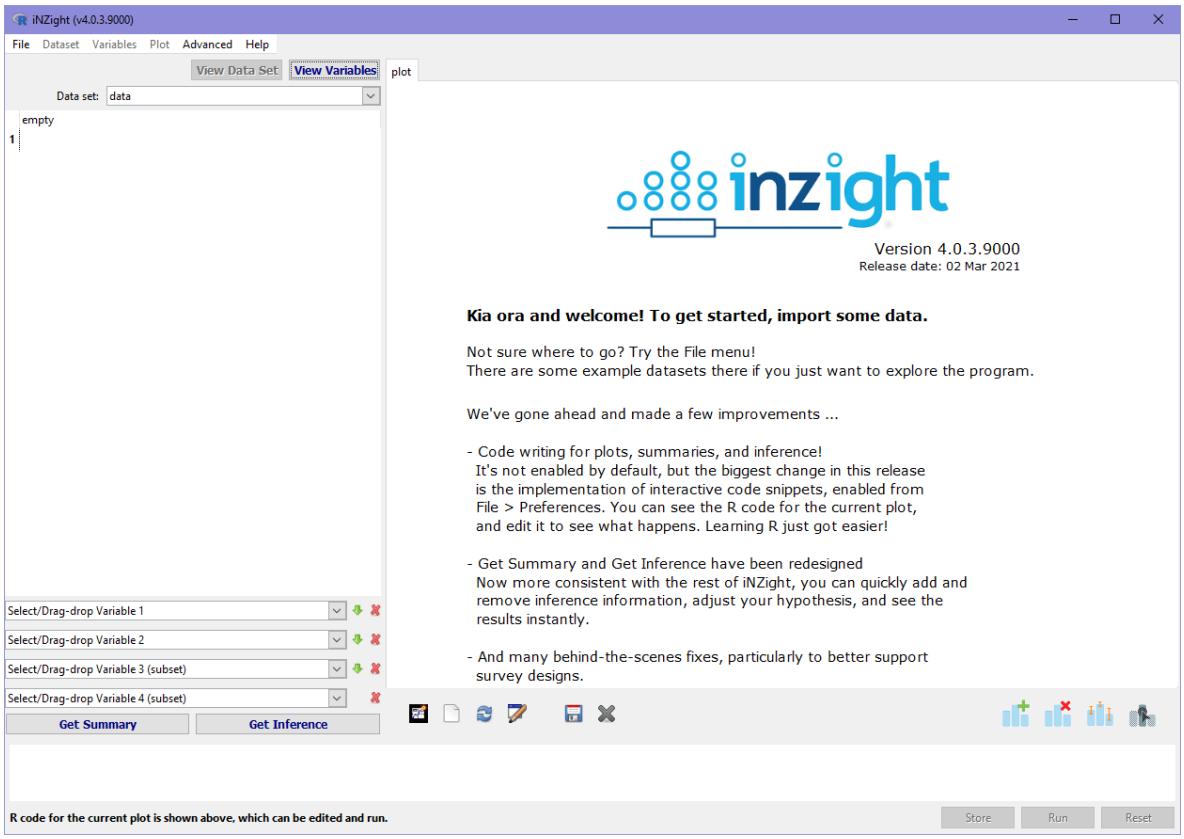

Figure 1: The **iNZight** GUI landing page presents users with a few controls. The screen will update once data has been loaded.

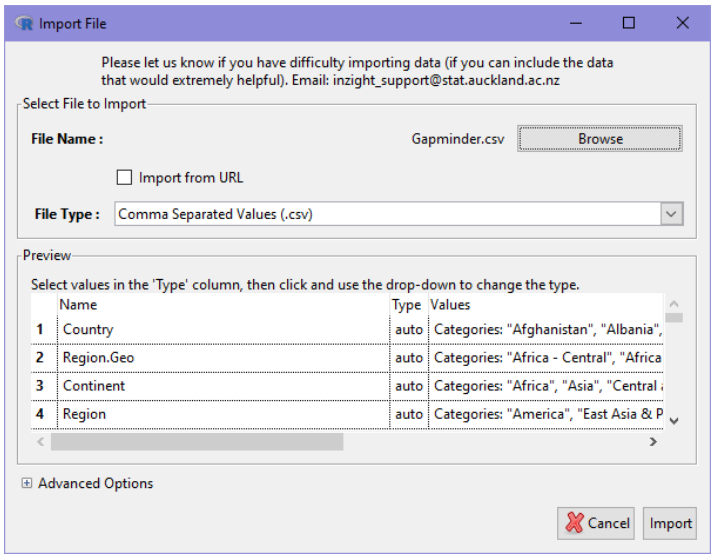

Figure 2: Load Data window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

dedicated to reading specific file formats. However, users must still know the correct package and function for their files. **iNZight** provides a simple Load Data window from which they can select a file to import. The software detects the file type from the file extension and attempts to read the file. If successfull, a preview is displayed for the user to check before proceeding to import the file. 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 CSV files). Namely, this allows users to override the default delimiter, for example in European countries where the semi-colon (;) is used (the comma is reserved as the decimal separator), as well as different encoding formats. The preview is updated when these options are changed, meaning users do not have to know specifically what they need, and can quickly trial-and-error until the preview looks right.

2.2. Creating graphs

Within **iNZight**, graphics are at the core of the user experience. The very first prototype included a 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, of 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 drag-and-drop of variables onto the Variable boxes (or using the drop down menus).

The first variable box is referred to as the *Primary Variable of Interest*. That is, this is the variable we want to know about, and how it is influenced by other variables. For example,

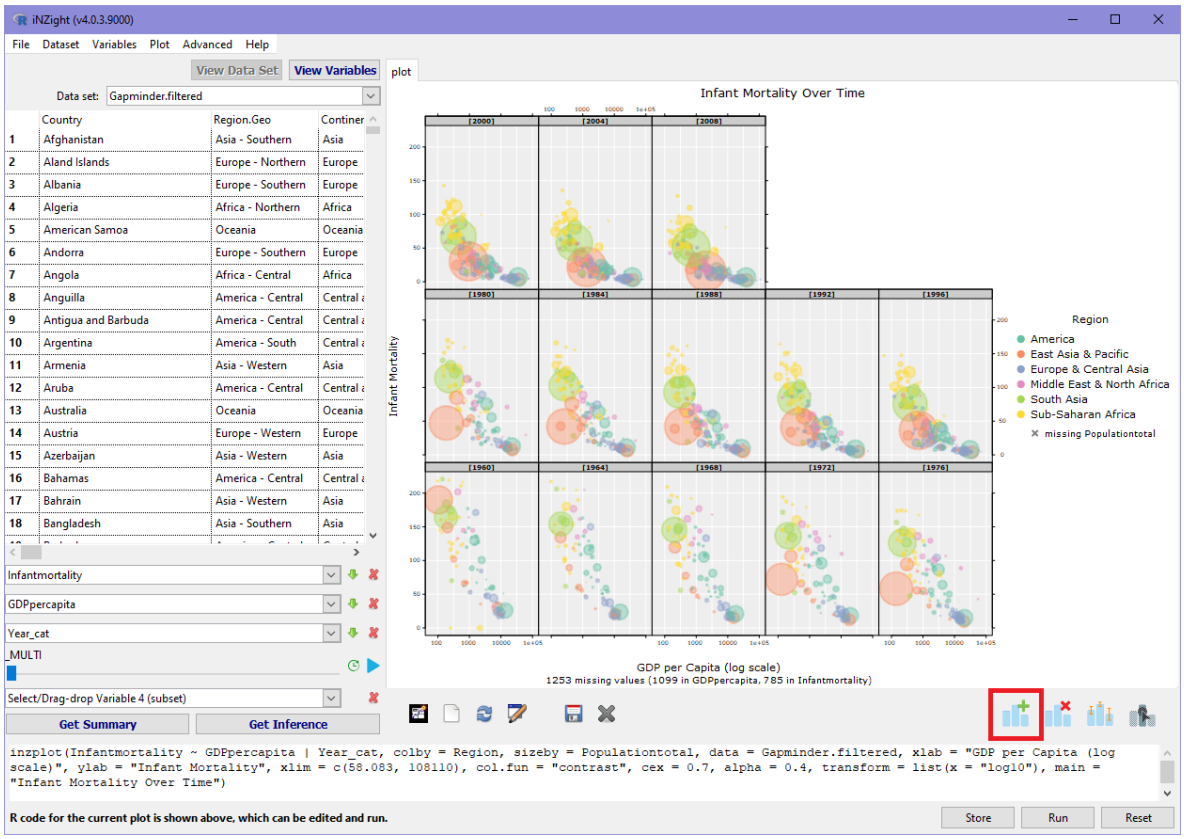

Figure 3: Demonstration of plot modifcations available from **iNZight**'s ADD TO PLOT menu. The ADD TO PLOT button, highlighted in red, opens a panel giving user control over colours, size, shape, labels, and much more.

if 'height' is chosen as Variable 1, and we want to know how 'height' changes with ethnicity or age. In the latter case, the result will be a scatter plot *with 'height' on the y-axis*. 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.

Finally, 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. 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 values, allowing them to explore "what happens if . . . ", and "what does this do?". The goal is to allow beginners to explore a dataset without being limited by having to learn how to do a wide range of actions. Figure 3 shows a graph produced by **iNZight** exploring the relationship between infant mortality and GDP, region, population, and year.

2.3. Summaries and inference

To supplement the visual graphics, **iNZight** also provides two textual output modes: *sum-*

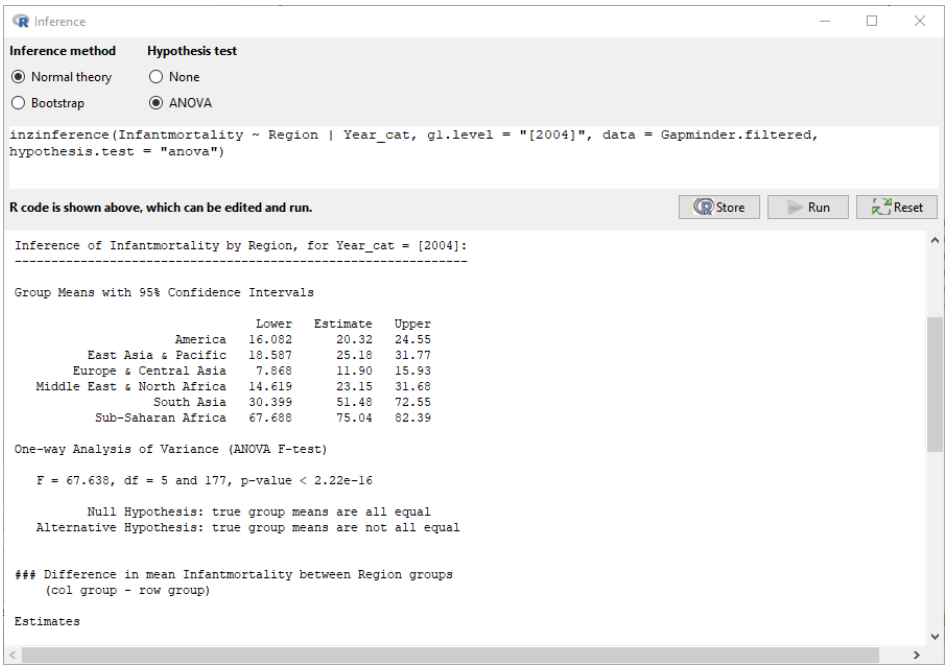

Figure 4: The INFERENCE window provides a selection of hypothesis tests for the chosen variables. In this case, these are `Infantmortality` and `Region`, so **iNZight** provides an ANOVA test.

maries and *inference*, accessed from the GET SUMMARY and GET INFERENCE buttons respectively. Summary information includes basic information about the chosen variables, including mean, standard deviation, quantiles, and so on, acting as a quick reference for values that are likely estimable from the graph itself.

The inference information provides confidence intervals for quantities such as means and proportions, and additionally provides 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

Usually the first thing researchers want to do when they first begin exploring a dataset is to create a set of exploratory graphs. However, it is often not possible to get the correct graphs from the raw data alone. Applying transformations and other modifications to the data can allow researchers to explore the data correctly, or explore 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 modifies individual columns.

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

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.

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 might yield the variable `Year.cat`, which makes the experience more exploration-friendly.

2.5. Special data types

Many data sets that our users will be exploring will be in 'tidy' format (Wickham and Grolemund 2017) (check citation), meaning 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 aspect of statistical education. These data sets require unique graphics to explore correctly, a task which **iNZight** has been extended to perform. Some of these are described here.

Complex survey designs

One of the most important data types for official statistics and research groups are complex surveys, and require information about the survey's structure to provide valid graphs, summaries, and inferences. **iNZight** handles survey designs behind the scenes, and only requires the user to specify the structure either manually (Figure 6), or by importing a special *survey design* file. Once specified, the user can forget about the survey design and use **iNZight** as normal: survey weights will be incorporated correctly into graphs and summaries, as well as data manipulation functions, using the **survey** (Lumley 2004) and **srvyr** (Freedman Ellis and Schneider 2020) packages behind the scenes.

iNZight handles two types of designs: stratified or cluster surveys, and replicate weight designs. In the former, information about the strata and clusters is contained within the dataset.

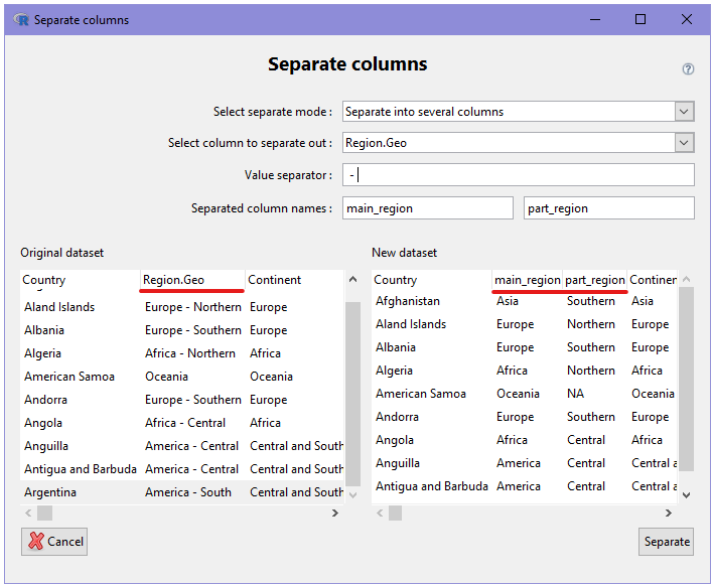

Figure 5: Here, the user is separating a column to create two new variable, with the preview displayed in the bottom-right. The relevent column names are underlined in red. The preview uses the first few rows of the data, and updates in real-time, reacting to changes the user makes, allowing them to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

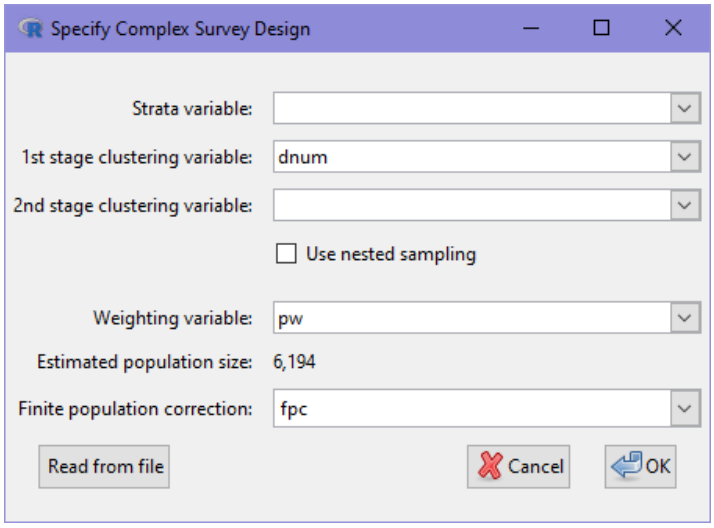

Figure 6: Users can specify survey design information manually by filling in the fields. These will then be used throughout the session.

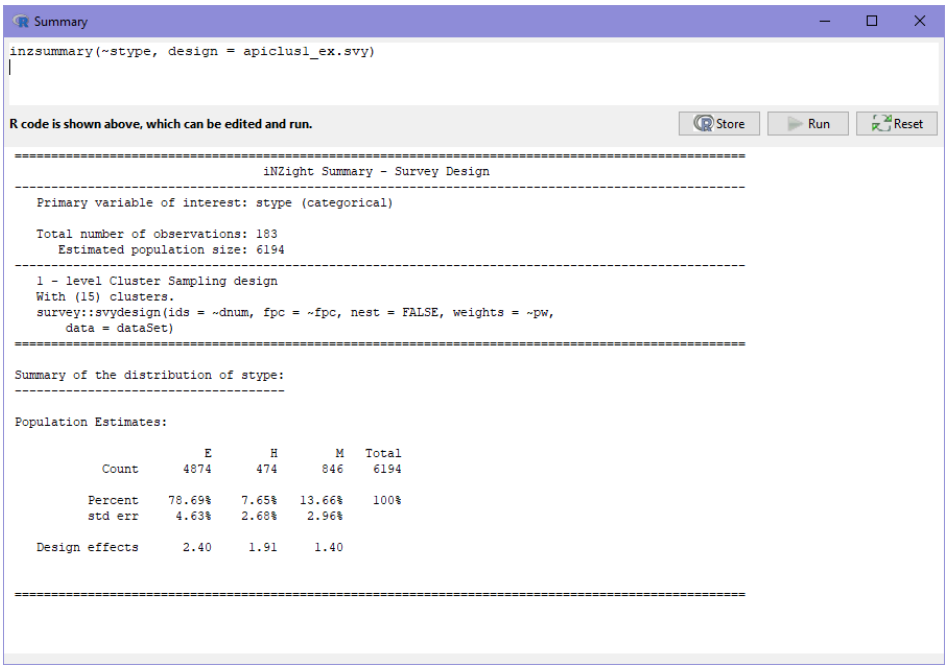

Figure 7: The SUMMARY window window provides simple summary statistics and, in the case of survey data, standard errors of these population estimates.

However, in the interest of confidentiality, some survey datasets use *replicate weights*, allowing variance estimates without exposing strata and cluster information (Lumley 2010). In addition to these, **iNZight** also provides functionality to calibrate surveys with data from other sources to reduce the estimation variances. Once again, this is performed once by the user and the used continually throughout the rest of **iNZight**.

The types of graphs available different for survey data. For a single numeric variable, a *histogram* is displayed 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 also. Bar plots remain the same. Summaries accessed from the GET SUMMARY button display the same information, but are summaries of the population values, and are given along with the standard errors of the estimates, as shown in FIGURE.

Time series

Another data type many beginners will come across is *time series*, and is taught in the final year of high school statistics in New Zealand (). The speciality of this data is that it must have one variable for observation times in a format that **iNZight** can understand, or can be specified manually if the user know the time information. Then a single variable can be displayed on the y axis as it changes over time, typically connected by dots.

iNZight provides capabilities for user to graph one or more time series on a graph, showing a smoother on each one. Additionally, for teaching purposes 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

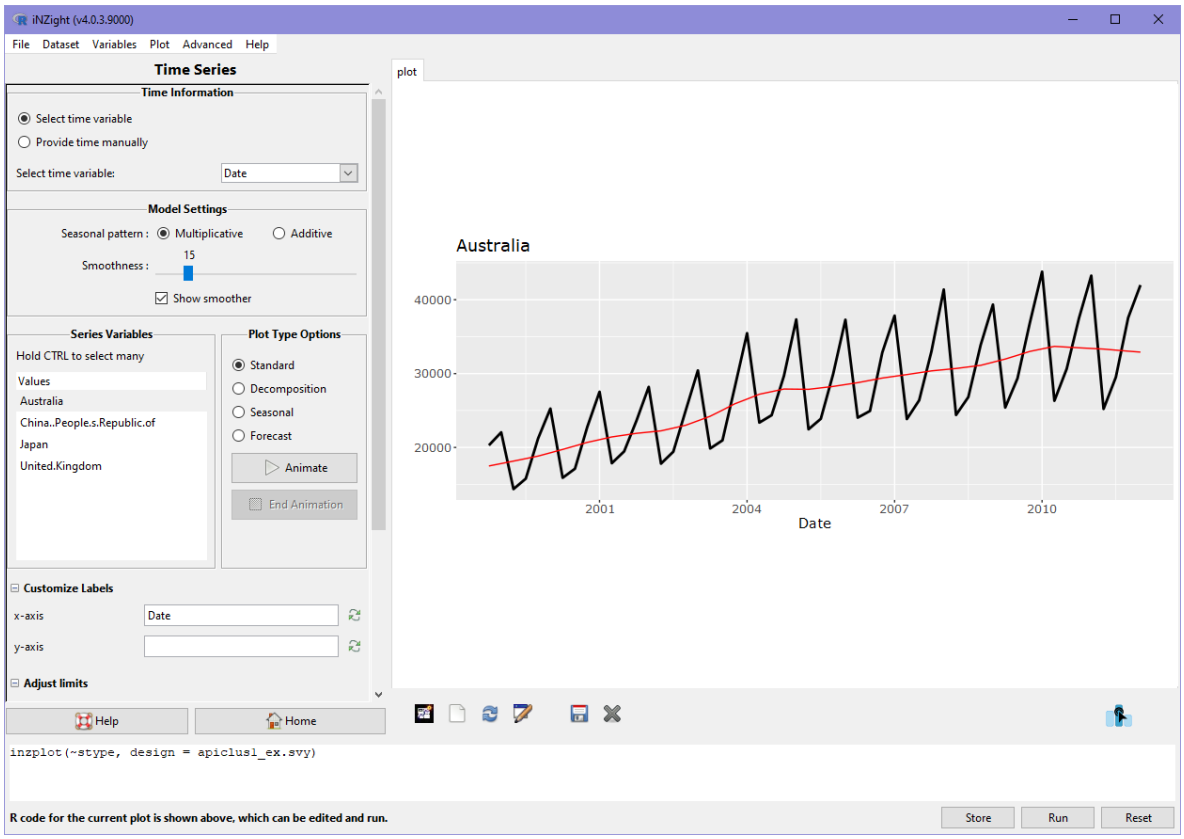

Figure 8: The time series module.

components combine to form the final series, and a simple Holt-Winters' forecast can be obtained similarly (Holt 2004; Winters 1960).

Figure 8 shows the time series module with New Zealand visitor arrivals data. The software automatically detects the `Time` column 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 VARIABLE list, one or more variables may be selected, and the appropriate graph drawn as chosen from the list on the right.

Maps

- another commonly seen data type
- either points on a map (e.g., a specialised scatter plot) or areas on a map, colour coded by variables
- **iNZight** can do both; regions requires data with location labels (countries, counties, regions, etc); points just a longitude and latitude variable
- lots of map-specific features: colour by region; aggregate time variable; plot mini plots over regions (e.g., barchart of categorical variable by region)

- the hard part is often figuring out shapefiles: **iNZight** lets user pick the type of map, and **iNZight** does the rest (with previews to help fine-tune)

Other types

- multiple response data
- multivariate data (addon module, Section 4)
- frequencies
- extensible to others too

2.6. Code writing for getting started with R

One feature of other R GUIs is the coding interface. For example, R Commander provides code for each action performed, and includes an executable script, while **Deducer** sits on top of the R console and integrates with the coding environment. However, these both require some level of R coding experience to use, as well as an understanding of the analyses to be performed. **iNZight**, however, is completely separate from the R console, proviing an interface-only experience for beginners. And yet, all of the code used by various actions exists and is available in the R Code script, allowing more advanced users interested in learning to code to get started.

Another feature is the inclusion of the R code box at the bottom of the interface, highlighted in FIGURE. This displays the code used to generate the current plot and, more importantly, can be edited by the user and run. 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. This script contains a history of all actions from importing the data through transformations and manipulations. The goal is to provide a record of what the user did, as well as something they can copy and paste into R and run themselves, editing where desired. This allows R learners to explore a dataset with a GUI tool and make the unique actions necessary, and then generate a script which they can build on and learn from. 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 **tidyverse** (Wickham *et al.* 2019) syntax, as this provides a simple introduction to R with many simpler and verb-like syntax for data wrangling, and is used the *R for Data Science* (Wickham and Grolemund 2017). To demonstrate the script produced by **iNZight**, Appendix A contains the script generated during the tour presented in this section.

3. Technical details

Figure 9: The reference class components of the **iNZight** interface, some of which are themselves made from several child objects.

The interface for **iNZight** is developed entirely within R, made possible by three main packages. **gWidgets2** (Verzani 2019) and **gWidgets2RGtk2** (Verzani 2020) provide a simple widget-based application programming interface (API) to building an 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+ (The GTK+ Team 2020) windowing system. Together, these packages provide a platform- and library-independent API for creating windows with uesr inputs from R.

The framework used to build **iNZight** is an object oriented programming (OOP) framework, which is well suited to GUIs, of which there are several within R. **iNZight** uses *reference classes*, which are also used by **gWidgets2**, to describe individual components of the interface. Each panel is a *class*, with individual buttons, methods (actions), and even smaller components which are themselves classes. OOP also allows for *inheritance*, which allows developers to describe a general class which can be shared to several related components, but which may have different layouts or methods. Figure 9 shows the **iNZight** GUI with some of the major class components annotated.

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* method,

which performs the primary action of the component. For example, the '`iNZFilterData`' class contains a `filter_data()` method. These main functions collect the user's input and pass them to one or more *wrapper* functions; in the case of '`iNZFilterData`', one of the functions is `iNZightTools::filterNumeric()`. A skeleton example of the Filter Data window class is shown below:

```
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$getActiveData())
            # ... construct GUI inputs ...
            # e.g.,
            variable <<- gcombobox(colnames(data))
            okbtn <- gbutton("Filter", handler = function(h, ...) filter_data())
        },
        filter_data = function() {
            filtered_data <- switch(type,
                "numeric" = iNZightTools::filterNumeric(
                    data,
                    var = variable,
                    op = operator,
                    num = value),
                ...
            )
            GUI$update_data(filtered_data)
        }
    )
)
```

In this oversimplified example, the user will be displayed a drop down `gcombobox()` to choose a variable to filter on. When they click the button, the data will be filtered and passed back to the main GUI. The filtering uses the `switch()` function to select the wrapper based on the users selection (not shown to save space).

Each major component has a similiar structure, and calls to various functions, many of which come from wrappers in other **iNZight*** packages. For example, plots are generated by calls to `iNZightPlots::inzplot()`, while data import is handled by `iNZightTools::smart_read()`. This function uses the file extension to guess the file type and load the data using the appro-

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.

priate methods. The wrappers enfore the desired structure, where the interface asks for the values of arguments to be passed, and keeps the data-specific logic out of the GUI.

A second advantage is that the individual 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. Here is an example of the result returned by `iNZightTools::smart_read()` (which can read from a remote URL):

```
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`).

While the GUI packages provide the structure of the visual GUI, it's the collection of R packages developed alongside **iNZight** that are the powerhouses of the program. The main reason for creating separate packages was to force the separation of interface and data logic, but also to allow 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.

- (not sure where to put this) diving deeper: data stored in *Documents*, each with a dataset/info about it; plots settings, variable choices, etc
  - users can switch between documents (data sets)
  - or merge them

3.1. Usage

At its core, **iNZight** is simply an R package that can be installed and run like any other, which is 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.

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

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

4. The add-on system

Most users will likely find all they need with 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 some data types (time series and maps, as examples). 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 opportunities. Rather than requiring each new datatype or method to be manually coded into **iNZight** by the developers, we have crafted an *Add on* system allowing anyone to create their own **iNZight** modules.

Installing existing add-ons is easy. Users can either use the Module manager to add, update, and remove modules from our add-on repository, or from a custom URL or 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.

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 shows a prototype of an upcoming Bayesian demographic modelling (Zhang *et al.* 2019) module which will be used by small groups to do small-area estimation. For example, estimation of 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.

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 publish 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 collections and names of libraries. However, the two dependencies are `xorg` and `gtk`, which are typically installed using the system package manager. For example, on the latest Ubuntu, 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.

5.2. Windows installer

The primary audience for **iNZight** is students new to statistics and likely computing too, and likely do not have advanced computer literacy that might be required by other GUIs to install and run the software (including R). To future 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 GUI. 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, which when launched triggers an update to 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 (Merkel 2014) is a development and deployment solution for developers to build, test, and share their projects. 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 on macOS and Linux to run the software without installing the system dependencies. The downside of this approach is that the result is not as smooth as a native application, 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 project the GUI onto the host's screen. More information can be found at `https://github.com/iNZightVIT/docker-inzight`.

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 the same tools as others, 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 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 (AWS) 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 is used to create the visual controls and perform reactivity events. A users data is stored on the server temporarily, and is only accessible from that user's session: it cannot be shared or access 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.

6. Summary and future work

- **iNZight** provides an easy-to-use GUI to simple exploration and visualisation tasks for statistics and data science students/beginners
- variable-first: doesn't require knowledge of techniques beforehand
- data manipulation techniques and other advanced data-specific features (e.g., surveys)
- code writing for those interested in migrating towards learning to code in R

- add-on system for easy expansion by course coordinators/researchers/R package maintainers who want to provide a simple interface to their package
- available on Windows and Linux

6.1. Future Work

- database connections (for larger datasets, including surveys, with database calculations where possible)
- translation system (flexible for any other languages)
- exploration of alternative GUI frameworks (gtk alternative) with better cross-platform support (i.e., macOS)

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, ..., 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

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.

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

## BETA WARNING: we're still working on making this as accurate
##               as possible, so please ...
##  - expect 'gaps' in the generated code (i.e., missing actions), and
##  - LET US KNOW if you think something's missing
##    (if you can give a minimal step-by-step to reproduce the problem,
##     that would be incredibly useful!)
##    email: inzight_support@stat.auckland.ac.nz

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

## 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(
        "Country" = as.factor(Country),
        "Region-Geo" = as.factor(`Region-Geo`),
        "Continent" = as.factor(Continent),
        "Region" = as.factor(Region),
        "Year_cat" = as.factor(`Year_cat`),
        "BodyMassIndex_M" = as.factor(`BodyMassIndex_M`),
        "BodyMassIndex_F" = as.factor(`BodyMassIndex_F`),
        "Cellphones" = as.factor(Cellphones),
        "Femalesaged25to54labourforceparticipationrate" = as.factor(Femalesaged25to54labou
        "Forestarea" = as.factor(Forestarea),
        "Governmenthealthspendingperpersontotal" = as.factor(Governmenthealthspendingperpe
        "Hightotechnologyexports" = as.factor(Hightotechnologyexports),
        "Hourlycompensation" = as.factor(Hourlycompensation),
        "Incomeshareofpoorest10pct" = as.factor(Incomeshareofpoorest10pct),
        "Incomeshareofrichest10pct" = as.factor(Incomeshareofrichest10pct),
        "Internetusers" = as.factor(Internetusers),
        "Literacyrateadulttotal" = as.factor(Literacyrateadulttotal),
        "Literacyrateyouthtotal" = as.factor(Literacyrateyouthtotal),
        "Longtermunemploymentrate" = as.factor(Longtermunemploymentrate),
        "Poverty" = as.factor(Poverty),
        "Ratioofgirlstoboysinprimaryandsecondaryeducation" = as.factor(Ratioofgirlstoboysi
        "Renewablewater" = as.factor(Renewablewater),
        "Taxrevenue" = as.factor(Taxrevenue),
        "TotalhealthspendingperpersonUS" = as.factor(TotalhealthspendingperpersonUS)
    ) %>%
    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",
    xlim = c(
        58.083,
        108110
    ),
    col.fun = "contrast",
    cex = 0.7,
    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
published by the Foundation for Open Access Statistics
MMMMMM YYYY, Volume VV, Issue II
doi:10.18637/jss.v000.i00

http://www.jstatsoft.org/
http://www.foastat.org/
Submitted: yyyy-mm-dd
Accepted: yyyy-mm-dd