Journal of Statistical Software

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

iNZight: A Graphical User Interface for Democratising Data with R

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, graphics, and 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 other statistical methods without any need to grapple with R code (but with opportunities to do so if desired). Two prominent examples are R Commander (Fox 2005, 2016) 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. Muenchen (2020b; 2020a) reviews these and other GUI interfaces to R, including BlueSky Statistics (BlueSky Statistics LLC 2021), Jamovi (Jamovi Project 2020), and **RKWard** (Rödiger *et al.* 2012). Each of these GUIs provide point-and-click interfaces that let users perform a set of procedures without the need to recall R function and argument names. However, they do require prior knowledge of the names of the underlying statistical procedures they want to invoke, what they do, and how to use them.

This paper introduces **iNZight**, another GUI system written in R through a collection of R packages, designed to bring data visualisation and analysis to a wider audience. The name **iNZight** is a play-on-words blending the word "insight" with the initials of our home country of New Zealand. The ability to explore, visualize, and analyse data without prior knowledge of statistical procedures is **iNZight**'s foundational design principle and the major point of difference from other R GUI interfaces. This principle has been applied to **iNZight**'s capability to appropriately, and seamlessly, handle data from complex survey designs, which other R GUI interfaces are not able to do (see Section 3.5.1).

A key difference between **iNZight** and other R GUIs available is its much broader and less technically-skilled audience, ranging from students through to analysts and researchers in community groups and government agencies, bringing with them a range of requirements and skill levels. To engage such audiences and democratise data anlysis skills, we employ

basic design principles that minimise demand for up-front knowledge—the names of tests and procedures, for example—and replace it with up-front information easily accessed through contextual dialogs. This led to **iNZight** taking on a "variable first" approach, in which users select the variables they are interested in and let the software automatically produce outputs based on a set of defaults. In this way, we facilitate and encourage rapid data exploration. Further details on the design principles of **iNZight** are given in Section 2.

The **iNZight** package has been adopted throughout New Zealand's statistical education programs from the year 9 school curriculum to postgraduate statistics courses. In addition, the combination of **iNZight**'s simplicity and powerful toolset make it a popular choice for research organisations and national statistics offices,[1] including the Australian Bureau of Statistics and StatsNZ. So the same free statistical software can be used from high school through to research environments, creating a simplified pathway for the development of future statistical analysts. This paper provides an overview of **iNZight**'s design principles (Section 2), main features (Section 3), technical details of its development (Section 4), an introduction to the *add-on* system (Section 5), and a description of the install process (Section 6).

2. Design principles for iNZight

The goal of democratisation is to make data accessible to a wider audience by removing barriers and restrictions to its access and use, thereby unlocking the ability to discover and apply the information contained within data. The existence of these barriers often comes down to the availability of time, funding, and the required skills. As we have noted in Wild *et al.* (2021), co-author Andrew Sporle is involved with statistical and health agencies in several small Pacific nations. Overwhelmed by the information needs of their own countries, combined with the reporting demands of key international agencies such as the United Nations, they face the triple burden of distance, small workforces, and insufficient funding. As a result, the national statistics offices in these countries find it difficult to recruit and retain people with good data science skills or afford to contract external expertise.[2]

[1] ANDREW: any island nation examples?
[2] A citation for this? Andrew?

Accessibility issues reach beyond national statistics offices. Indigenous nations and communities, such as those associated with the Global Indigenous Data Alliance,[3] seek to own, govern, and apply data for the self-determined well-being of Indigenous Peoples (Walter *et al.* 2020). Indigenous nations and governance groups not only face resource and skill constraints, but indigenous data sovereignty and governance requires maintaining control over indigenous data resources. **iNZight** not only provides an accessible data analysis and graphics tool, but does so without the need to upload data beyond a user's own data environment—possession and control over the data are maintained.

The situation is very similar for subject-matter researchers from many areas who have projects with potential societal value, but who lack funds or access to technical skills. The unmet data-science needs of those lacking in money and data-education can be every bit as real and important as those who have more. Since there will never be enough altruistic, skilled volunteers to meet all these needs, it is important to enable more people to do more for themselves in statistically robust ways. And while the R GUIs mentioned in Section 1 do this at some level, some key elements have been overlooked that **iNZight** can now address.

For groups like those described above, generalists need to be empowered to do things that currently only specialists can do. Typically, these generalists are already busy—and thus time-poor—and they have very little data-related education. Addressing the lack-of-knowledge problem with up-front education runs straight into the lack-of-time problem. Additionally, generalists tend to do many different things and to work in specific processes only infrequently, so working around the rapid fading of memories of how to do things and what they mean is also a major factor. This collides with the "problem of names": with programming and most of the existing data analysis GUI systems, you basically cannot do anything until you know what you want to do and know/remember its name. This is a significant barrier to getting started and also results in significant time-losses getting back up to speed after a period of inactivity and a subsequent loss of familiarity. Even students taking service courses in statistics and data science, particularly those majoring in other subjects, experience long time delays between what they learn in class and almost anything they end up applying in real

[3] https://www.gida-global.org/

life. To circumvent these difficulties and seriously empower these groups, we need tools that reduce the demands for up-front knowledge and place much less reliance on leaky memories. The **iNZight** project is addressing these problems through some simple design principles:

1. require as little user input as possible through automation and defaults;
2. use context to display relevant options based on user choices to reduce dependence on up-front knowledge;
3. guide users through complicated procedures by asking for one piece of information at a time;
4. display alternative options only when applicable to the data type or task.

The basic mode of **iNZight** provides visualizations and analyses for rectangular data with variables in columns and observations in rows. **iNZight** uses a "variable-first" design in which users drive the software by first choosing variables rather than procedures. Users assign roles to variables with immediate responses determined by variable-type and defaults (variable-types currently recognized are: categorical, numeric, and date-time). The underlying metaphor is, *"Tell me about ..."*, so tell me about a variable—or a relationship between two variables—either alone or subsetted/facetted by other variables. Here, *"Tell me about"* is really *"Show and Tell"* because what is delivered instantly is graphics because we believe graphics are the most accessible, information-rich artefacts for broad audiences, and also that people are less likely to do silly things when they look at their data first.

To obtain numeric information, you explicitly ask for it by clicking Get Summary or Get Inference. The underlying metaphor for both is, *"Give me the types of information analysts generally want to see in a situation like this"*. So Get Inference gives users their analyses of variance, chi-squared tests, regression panels, and other available tests in situations where they are appropriate, accompanied by sets of relevant confidence intervals. Users do not have to know or remember what to ask for or how to ask for it, demonstrating one of the many barriers knocked down by **iNZight**—that of entry for beginners and re-entry for users returning after a period of non-use. After-the-fact information concerning *"How can I read*

this and what does it mean?" has compelling relevance when you have output in front of you. The up-front knowledge required by **iNZight** in its basic mode is simply some high-level familiarity with rectangular data, variables, and the ability to identify situations where users might want to override a variable-type default (e.g., numeric codes used as group labels).

In the basic mode described above we have automated everything by using defaults and delivering immediate results. This, however, begs the question, *"How else can I look at this?"* A plot-type selection box allows scrolling (with a mouse wheel or arrow key) through plots from all the applicable graph-types in the Financial Times Visual Vocabulary,[4] with some additions. Use of defaults also begs, *"How else can I do this?"*, for example by making applicable alternatives available (through dropdowns or other controls) in place of the defaults provided. Inferences, for example, may be based on normal theory or bootstrapping, and a switch can turn on epidemiological versions of outputs (for example odds- and risk-ratios) when appropriate. Options for plot enhancements are extensive including: information-adding mechanisms like coding of additional variables using colour, size, and symbol; adding trend lines and other inferential mark-up; identification and labelling of points; motion (playing through a set of faceted graphs); interactivity; and many additional modifications that might be desired for aesthetic reasons.

Another feature of **iNZight** is its code-writing functionality, which it shares with other GUIs, but in a less prominent manner. For most users, no R code is ever seen; however, each action invoked by the user runs some R code, which is appended to the R script, with some select instances of live-code display and editing available (more details on this in Section 3.6). Originally the code-writing functionality was developed as a way for students to become familiar with R code after already developing basic data visualisation, exploration, and analysis skills: they use the tool as normal, but can now obtain the R code used to perform the analysis, and begin editing and running the code themselves in R rather than having to start from a blank script. Furthermore, this same feature makes **iNZight** a powerful research development tool by allowing users to quickly explore a dataset while simultaneously creating an R script that can form the basis of a robust, reproducible workflow in a research setting.

[4] http://www.vizwiz.com/2018/07/visual-vocabulary.html

iNZight's high-level, variable-driven requests accompanied by instant results facilitates and encourages rapid data exploration. By significantly reducing the barriers to access, performing data-analysis tasks become accessible to a diverse range of individuals and organisations. In doing so, **iNZight** is democratising data, a highly valuable resource in our modern digital world.

3. A tour of iNZight's features

iNZight was originally built as a tool for high school statistics students, and was later incorporated into introductory statistics courses in New Zealand. It has since matured and can now function as a rapid research development tool for community and government research groups. The minimal interface provides instant feedback to user actions, allowing users to intuitively explore their data. The *variable first* paradigm makes it easy to learn new statistical concepts—such as hypothesis testing—by focusing on the output first, and consequently makes it very easy to pick up after a period of non-use, a typical scenario for generalists within organisations. Ease-of-use is achieved through familiar controls such as *drag-and-drop*, *drop-down* selection, and *slider bars*. Using these controls, users select the variables they are interested in and the software reacts instantly based on variable type and defaults. The simplest way to explore **iNZight**'s features is by demonstration.

3.1. Importing 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). Fortunately, there are thousands of R packages on the Comprehensive R Archive Network (CRAN), amongst which are some dedicated to importing most of the common (and indeed many 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 Import Data window from which users need only select a file to import: by default the software detects the file type from the filename extension and

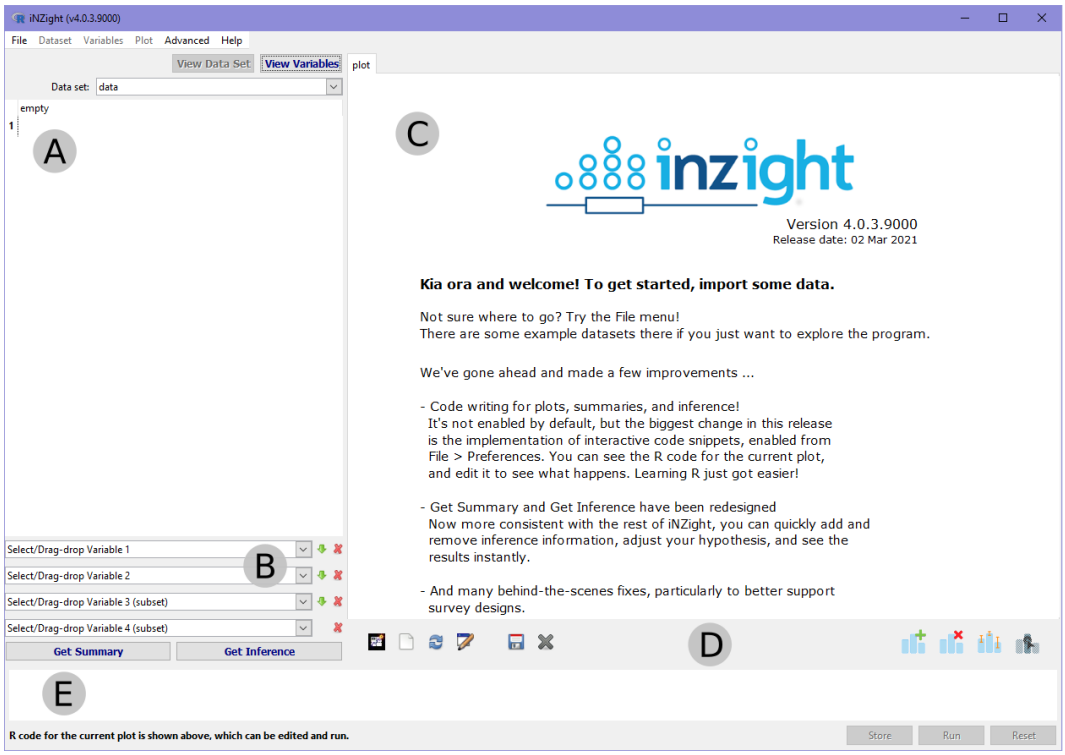

Figure 1: The **iNZight** GUI landing page presents users with a few controls. The labeled areas are: (A) the active data set is displayed prominently; (B) variable control boxes provide users either drag-and-drop from (A), or select from dropdowns; (C) graphs are displayed in the graphics window; (D)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

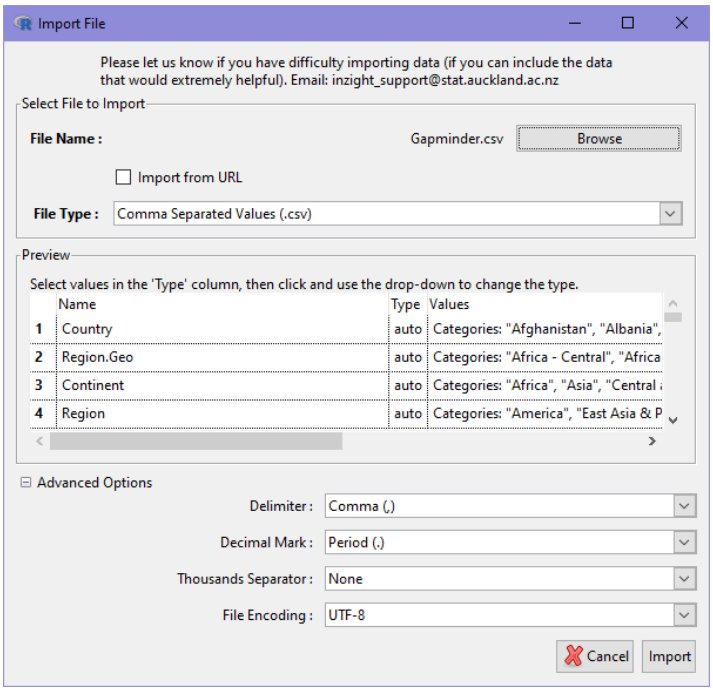

Figure 2: Load Data window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

reads the file if the format is supported. If a dataset is available publicly on the internet, users can also provide a URL from which **iNZight** will import the data instead.

Currently, **iNZight** supports files in comma separated values (CSV), tab-delimited text, 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 Import Data window for a CSV file. The Import Data window has an Advanced Options section, expanded in Figure 2, for CSV and delimited-text files 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** supports numeric, categorical, and date-time formats—with more on the way.

iNZight also supports copying and pasting data from a spreadsheet to make things even easier for beginners.

3.2. 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 from a small-to-moderate sized dataset is visualised (by default) with a dot plot, while a single categorical variable defaults to 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?"* and *"What is it telling us?"*. This, together with the ability to scroll quickly through a set of alternative applicable graphs, also means experienced users can very quickly look at a variety of variables and 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 relationships between those variables and 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 by default 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 (seen in Figure 3), 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 with approximately equal numbers of observations in each.

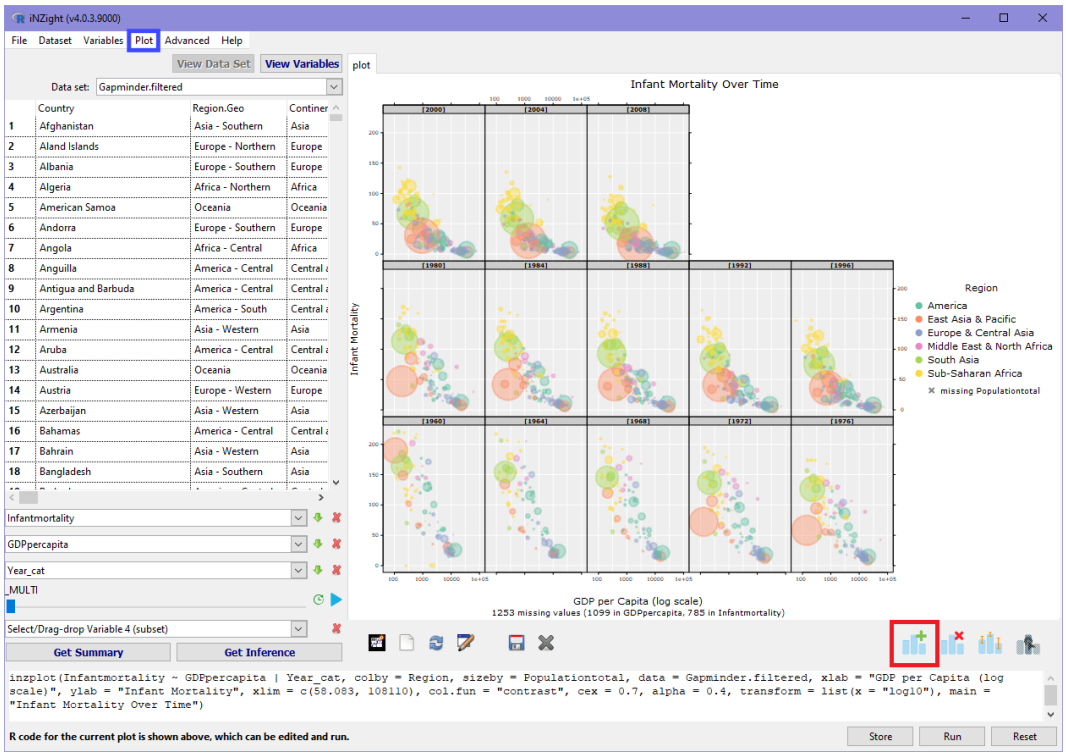

Figure 3: Demonstration of plot modifications available from **iNZight**'s ADD TO PLOT menu. The ADD TO PLOT button, highlighted in red, opens a panel giving user control over colours, size, shape, labels, and much more. This can also be accessed from the plot menu, boxed in blue.

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 button in the PLOT TOOLBAR (boxed in red in Figure 3). Users can choose from a selection of alternative plot types based 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 more about what the software can do 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 (subsetting/faceting) using the

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/independence
[5] Additionally includes epidemiological output such as odds and risk ratios.

Gapminder dataset.[5]

3.3. Summaries and inference

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. As previously noted, the underlying metaphor for both is, *"Give me the types of information analysts generally want to see in a situation like this"*. 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 summaries for each subset are provided.

The inference information includes estimates, confidence intervals, and any applicable p-values for quantities such as means, proportions, and their differences. For performing hypothesis testing, **iNZight** displays a set of tests applicable to the chosen variable(s), as shown in Figure 4 (the full list of tests currently available in **iNZight**'s basic mode 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.

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

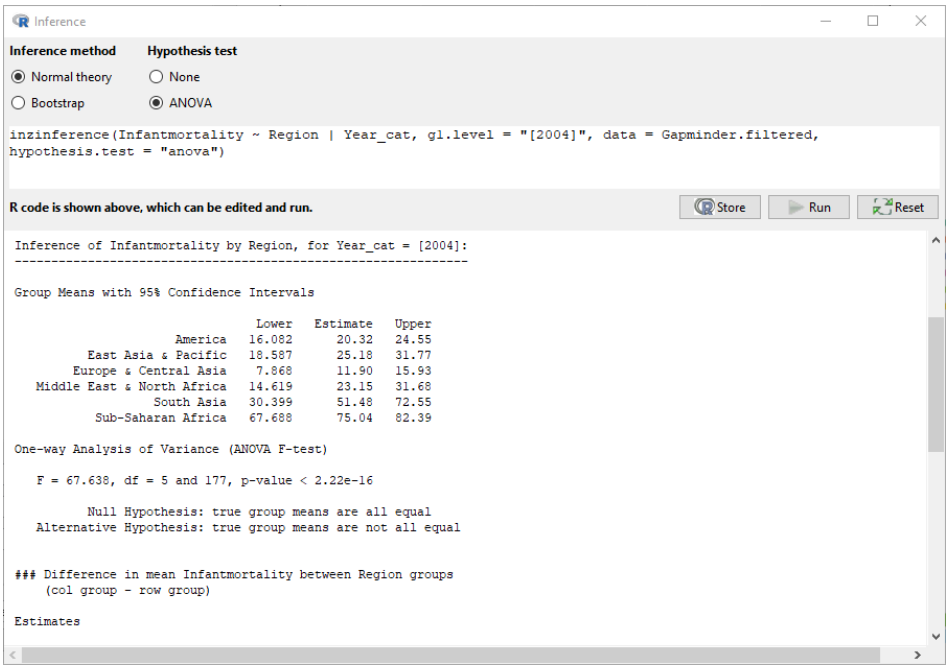

Figure 4: The INFERENCE window provides a selection of hypothesis tests for the chosen variables. In this case, these are `Infantmortality` (a numeric variable) and `Region` (categorical with six levels), so **iNZight** provides an ANOVA test.

3.4. Data wrangling

Researchers typically start a new analysis by creating a set of exploratory graphs, as described in Section 3.2. 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 do things like filtering by value, converting from *wide* to *long* form, or joining two datasets together. In most cases, the interface evolves from

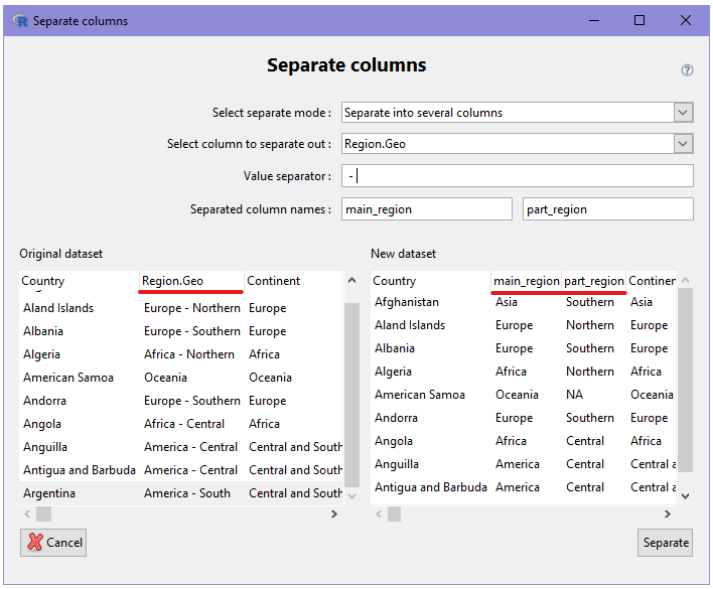

Figure 5: Here, the user is separating a column to create two new variables, with the preview displayed in the bottom-right. The relevant column names are underlined in red. The preview uses the first few rows of the data, and updates in real-time, reacting to changes the user makes, allowing them to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

top-to-bottom as the user first chooses, for example, the variable to be worked on, with subsequent inputs tailored to previous choices, thus guiding users through the procedure. At the bottom of many data manipulation windows is a preview of the data after transformation, as demonstrated in Figure 5, once again allowing users to play with inputs to get the desired output. 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 is `Year`), or categorical variable levels can be renamed, reordered, or combined. Users can also create new variables based on others in the dataset, and rename or delete existing variables. **iNZight** creates a *new* variable for each action—for example, converting `Year` to categorical yields a new variable named `Year_cat` by default—which makes the experience more transparent and more exploration-friendly. Appendix A.2 gives a list of available variable manipulation methods.

3.5. Additional facilities for 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 variables. There are, of course, several types of data that are an important component of statistical analysis, or are commonly encountered by those new to the field. These types of data often require specialist graphics or handling—often both using specialist software or R packages. Here we provide several examples of alternative data types that **iNZight** can handle.

Complex survey designs

One of the more important data types demographers and researchers using official statistics encounter is data from complex surveys, which requires 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. Data producers can distribute the design file with their datasets to ensure that users conduct valid analyses of their data. Once specified to **iNZight**, the user can forget about the survey design and use **iNZight** as normal: survey weights and other design information is incorporated correctly into graphs, summaries, and data manipulation functions using the R packages **survey** (Lumley 2004) and **srvyr** (Freedman Ellis and Schneider 2020) behind the scenes. The R code (Section 3.6) for specifying the survey design is included in **iNZight**'s code history.

iNZight handles stratified as well as one- and two-stage cluster surveys. Additionally, **iNZight** supports *replicate weight designs* in which survey design information is replaced by a set of replicate weights that allow surveys to be analysed correctly without exposing confidential or private information contained within the design variables (for example, clusters). Instead, sets of "replicate weights" are provided, a mechanism that still permits valid variance estimation (Lumley 2010). **iNZight** can also calibrate surveys with population data from other sources to reduce the variances of estimates. Once again, this information is specified once, either by the user or a survey design file, and is subsequently used throughout the rest of the analysis.

The types of graphs available differ for survey data, as they now represent a picture of the *population*, and not simply a sample. For a single numeric variable, a *histogram* is displayed

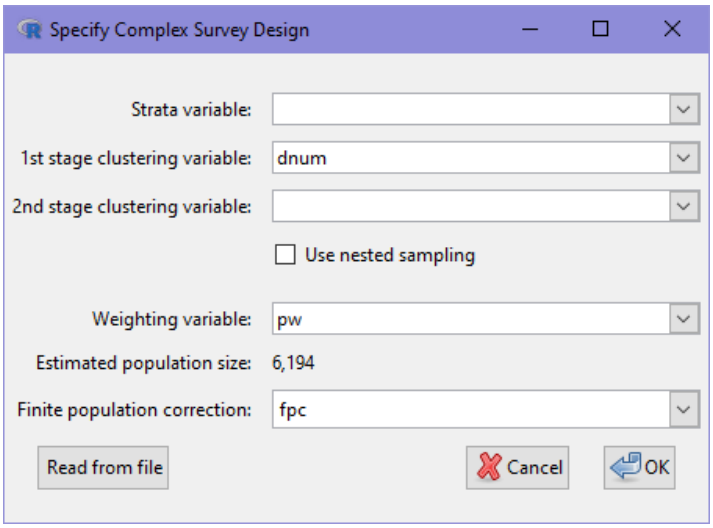

Figure 6: Users can specify survey design information manually by filling in the fields. These will then be used throughout the session thereafter where appropriate.

by default instead of a dot plot. For two numeric variables, a *bubble plot* is used, which is a scatter plot with points sized by the weights of respective observations; alternatively, this can be displayed as a *hex-bin* plot, which can be particularly effective when there are large numbers of observations. Bar plots are still used by default for categorical variables in surveys. Summaries display the same information as before (Section 3.3), 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 by using the survey equivalents from the **survey** package.

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 observations 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 on each.

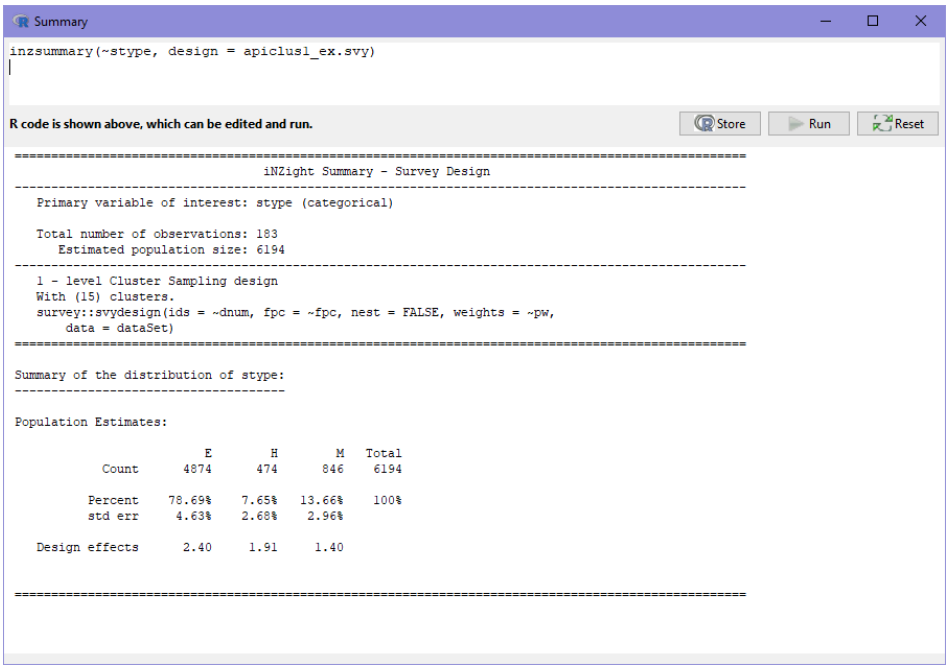

Figure 7: The SUMMARY window provides simple summary statistics and, in the case of survey data, standard errors of these population estimates.

Additionally, the series can be decomposed into trend, seasonal, and residual components (using seasonal-trend decomposition using LOESS). 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 to be displayed on the graph can 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

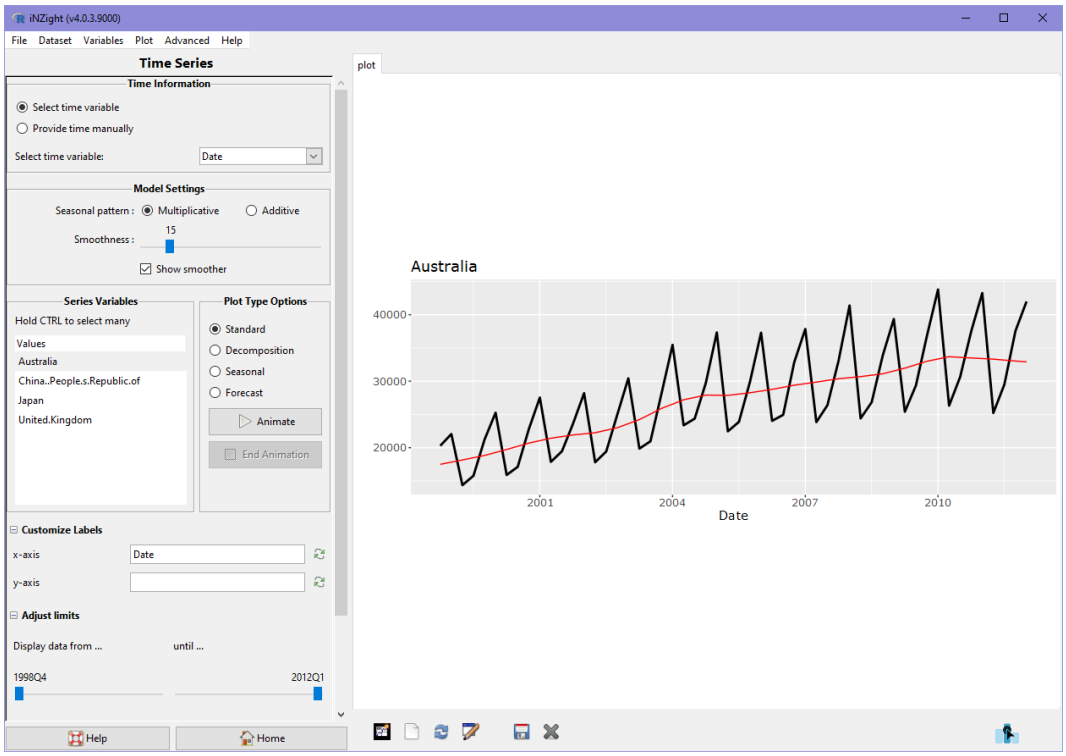

Figure 8: The time series module.

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 (for example: countries, provinces, states). The functionality within the maps module calls wrapper functions from the **iNZightMaps** package (Barnett and Elliott 2020).

For point-based observations, points are overlaid 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, 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,[6] 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 a definition of the boundaries of the areas contained in the data. For example, New Zealand can be divided into regional councils

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

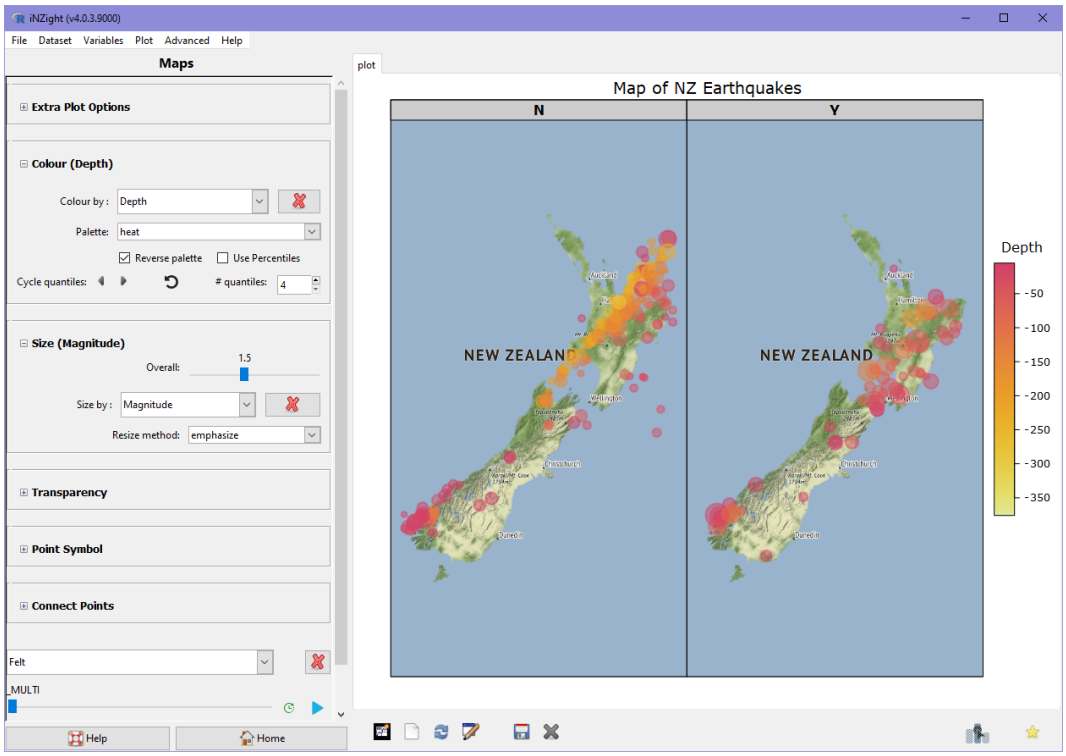

Figure 9: The maps module showing New Zealand earthquakes sized by magnitude, coloured by depth, and subset by whether or not they were felt.

(Auckland, Otago, etc), which have physical boundaries that can be described by an external shape file. The Maps module in **iNZight** lets users choose the type of regions contained in the data, and proceeds to match labels between the dataset and map using several matching techniques: for example, countries may be coded using their conventional names, official names, or three-letter codes ("New Zealand" versus "NZ" versus "NZL"). Once the initial set-up is complete, users can choose variables to graph, which are then visualized on the map using colour, points, and other techniques. Often users are interested in regional trends over time, so **iNZight** automatically detects this and can animate maps over time or display trends as *sparklines*.[7]

Other data types and features

Besides these examples, **iNZight** has several other modules that allow more complex statistical methodology or support other special data types. *Multiple Response* data arises from

[7]Cite these sparklines? or an example??

"*Choose all that apply*" type survey questions, and need their own method of graphics to explore adequately. There is also a multivariate data add-on module that can perform and visualise multivariate statistical methods such as principal components analysis and non-metric multidimensional scaling. 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 updated automatically as users add and remove explanatory variables, and a range of residual plots are available to explore and help users to quickly fit a model with any necessary transformations.

Besides the examples listed above, **iNZight** has an add-on system (Section 5) 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 publicly or privately.

3.6. Code writing for getting started with R

[8] 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 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, applying transformations and manipulations, to any plots and summaries the user chose to save. This provides a record of what the user did which can be saved and run in R itself,

[8] Move/mention up front when R Commander / Deducer first introduced?

with edits if desired.[9] This lets users explore a dataset with a GUI tool to quickly start an analysis 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.

The code writing features of **iNZight** are described in more detail in Section 3.2.2 of Burr *et al.* (2021) together with the educational imperatives it facilitates. In addition, extremely high-level functions in the **iNZightPlots** package (Elliott *et al.* 2020) have been written to facilitate quite a few of the low-human-memory features of **iNZight**'s GUI environment in a coding environment (see Sections 3 and 4 of Burr *et al.* 2021).

4. Technical details

iNZight's interface is written with R using the three support packages **gWidgets2**, **gWidgets2RGtk2**, and **RGtk2**. **gWidgets2** (Verzani 2019) provides a simple widget-based application programming interface (API) to building a cross-platform interface with R, with support for **Tcl/Tk** (Raines *et al.* 1999), **Qt** Nokia Corporation (2021), and **GTK+ 2.0** (The GTK+ Team 2020). We chose GTK2, as it was the most feature rich and—at the time—had the best cross-platform support (see Section 6.1). The GTK2 binaries are accessed through R using the

[9] Remove repetition.

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

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.

4.1. Component design

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 drives **iNZight**'s add-on system (Section 5). 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.

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 are also *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 performs the desired action. 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 similar 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 receiving user input and displaying the output.

4.2. Code-writing wrapper functions

Another advantage of having components calling external functions is that the wrapper functions can attach 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 3.6, all while keeping the GUI and data-oriented code separate. Here is an example using `iNZightTools::smart_read()`:

```
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 processing and variable manipulation.

```
R> library("iNZightTools")
R> data <- smart_read("nls.dta")
R> print_code(data)

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

The `iNZightTools::code()` function can also be used to get the R code attached to the resulting object, allowing beginner R users 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 examine and modify the underlying code returned by `smart_read()`. In this way, novice R users can read a wide variety of datasets, and can very quickly adapt the provided code to fit unique situations by reading the documentation for the supplied function, rather than having to do the time consuming job of searching for the correct package and starting from scratch.

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. Additionally it allows parallel development of a separate Shiny (Chang *et al.* 2021) interface (Section 6.4) 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 beginner R users to use standalone (Burr *et al.* 2021, Sections 3 and 4).

4.3. Usage

At its core, **iNZight** is an R package that can be installed and run like any other (see Section 6). 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. The `data` argument could also be used 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 of the **iNZight** GUI 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, developers 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 the current data, while the second sets the *Variable 1* drop-down (`V1box`) value to `height`.

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

[1] 500  10

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

The second line would trigger the plotting of 'height'.

5. 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 into 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,[10] a custom URL, or a local file. In all cases, the file is downloaded to the `modules` directory contained in the base **iNZight** installation folder, although users can also place files in this directory manually. All valid files in this directory are displayed as menu items 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 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 module based on the work by Zhang *et al.* (2019), 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.

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

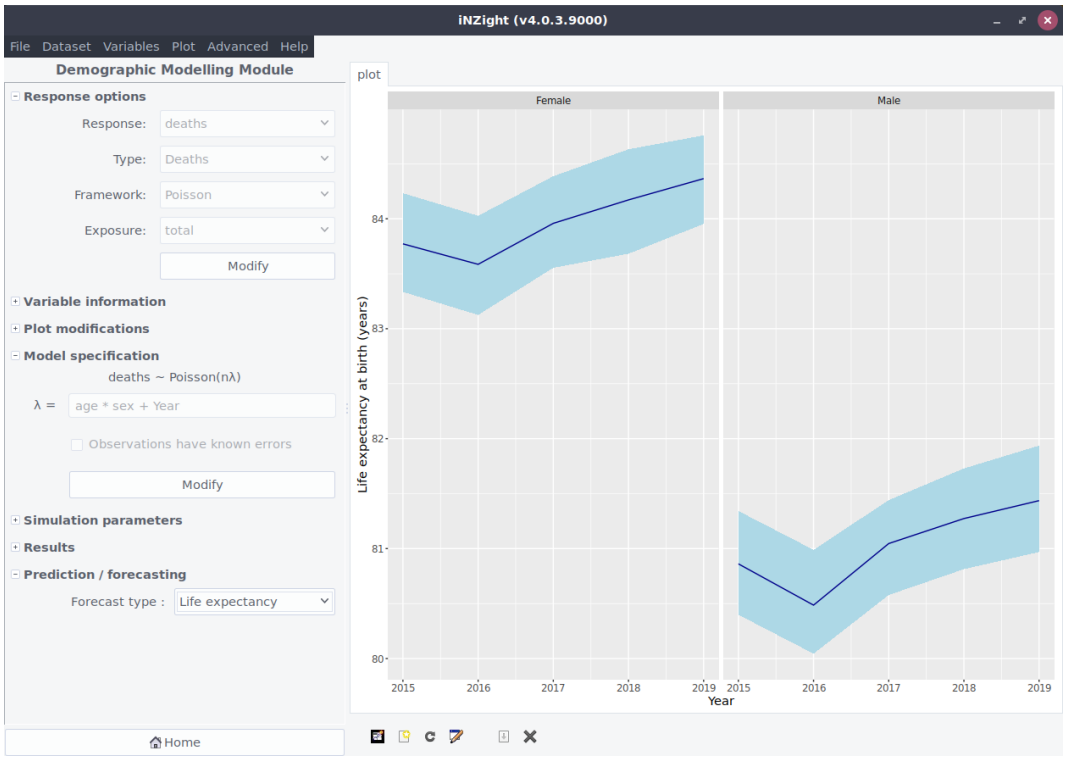

Figure 11: The prototype for a new Bayesian demographic modelling module for **iNZight**. In this example, life expectancy is estimated from death data.

Pull some more details from dedicated report.

6. 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.

6.1. 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 the binaries are no longer available from 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 6.2) under Wine.[11]

6.2. Windows installer

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 archive 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`

[11] `https://www.winehq.org/`

file which automatically loads the **iNZight** package and launches the interface. It also hides the R console, so users are presented with just the GUI. When started from this 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 just deletes the folder and any shortcuts.

6.3. 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 end 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`.

6.4. Online shiny version iNZight *Lite*

In recent years, many schools have adopted tablets or Chromebooks instead of 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**, named **iNZight** *Lite*, that uses **shiny** (Chang *et al.* 2021) as the GUI framework instead of **GTK**.

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.

Our 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 example it is possible to store datasets on a server linked to URLs so that *Lite* is launched with the chosen dataset already loaded. For example, `https://lite.docker.stat.auckland.ac.nz/?url=https://inzight.nz/testdata/nhanes.csv&land=visualize`. An example of 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 ...

7. 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. By taking a *variable first* approach, users do not need to 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 effectively 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 and get the output they are after. Many users may want to learn to code with R, and **iNZight** includes some simple tools for helping that transition: 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 in addition to **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. These audiences, in return, gain access to methods previously inaccessible to anyone not well-versed in R coding, thereby democratising complex but crucial methods such as Bayesian demography.

7.1. 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 a desire to develop a fully cross-platform application so users from all backgrounds can take advantage of **iNZight** to explore their data.

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 NZ; NZ 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.

BlueSky Statistics LLC (2021). *BlueSky Statistics*. www.blueskystatistics.com.

Burr W, Chevalier F, Collins C, Gibbs AL, Ng R, Wild CJ (2021). "Computational Skills by Stealth in Introductory Data Science Teaching." *Teaching Statistics*, **43**(S1), S34–S51. doi:10.1111/test.12277.

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.

Elliott T, Soh YH, Barnett D (2020). ***iNZightPlots**: Graphical Tools for Exploring Data with **iNZight***. Available from https://CRAN.R-project.org/package=iNZightPlots.

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.

Fox J (2016). *Using the R Commander: A Point-and-Click Interface for R*. Chapman and Hall/CRC. ISBN 9781498741903.

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.

Muenchen RA (2020a). "R Graphical User Interface Comparison." http://r4stats.com/articles/software-reviews/r-gui-comparison/. Accessed: 2021-09-27.

Muenchen RA (2020b). "Reviews of Data Science Software." http://r4stats.com/articles/software-reviews/. Accessed: 2021-09-27.

Nokia Corporation (2021). ***Qt**: Cross-Platform Software Development for Embedded & Desktop*. https://www.qt.io/.

Jamovi Project T (2020). *Jamovi*. https://www.jamovi.org.

Raines P, Tranter J, Oram A (1999). ***Tcl/Tk** in a Nutshell*. O'Reilly Media, Inc. ISBN 9781565924338.

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/.

Rödiger S, Friedrichsmeier T, Kapat P, Michalke M (2012). "**RKWard**: A Comprehensive Graphical User Interface and Integrated Development Environment for Statistical Analysis with R." *Journal of Statistical Software*, **49**(9), 1–34. doi:10.18637/jss.v049.i09.

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.

Walter M, Kukutai T, Carroll SR, Rodriguez-Lonebear D (2020). *Indigenous Data Sovereignty and Policy*. 1st edition. Routeledge. doi:10.4324/9780429273957.

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.

Wild CJ, Elliott T, Sporle A (2021). "On Democratizing Data Science: Some iNZights Into Empowering the Many." *Harvard Data Science Review*. doi:10.1162/99608f92.85206ff9. Https://hdsr.mitpress.mit.edu/pub/8fxt1zop, URL https://hdsr.mitpress.mit.edu/pub/8fxt1zop.

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 3 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