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) allow inexperienced users to gain access to these activities by removing the need for coding. However, currently available R GUIs require some degree of experience with R and familiarity with many statistical concepts. **iNZight**, a GUI written in R, enables both students and researchers to interact with and explore data without the need for code and only minimal statistical knowledge. Generalists in government organisations are also able obtain official statistics for reporting purposes. The tool is designed to be easy to use, with intuitive controls, clever defaults for common tasks and other design features to lower the barriers to entry. **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 everything in between. Several packages provide graphical user interfaces (GUIs), allowing users to interact with R without grappling with code—but with opportunities to do so if desired. These GUIs em-

power a broad audience to create graphs, test hypotheses, and access other statistical methods that were previously accessible only to R coders. Two prominent examples are R Commander (Fox 2005, 2016) and **Deducer** (Fellows 2012). R Commander is a self-contained interface that displays, writes, and runs user-editable R code. **Deducer** extends the R console with menus that open GUI interfaces for various statistical methods. Muenchen (2020a; 2020b) provides a detailed review of these and other GUI interfaces to R, including BlueSky Statistics (BlueSky Statistics LLC 2021), Jamovi (Jamovi Project 2020), and **RKWard** (Rödiger *et al.* 2012). These tools have point-and-click interfaces that let users perform a set of procedures without the need to recall the 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 consisting of a collection of R packages to enable bringing data visualisation and analysis to an even 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 begin exploring, visualizing, and analysing data without prior knowledge of statistical procedures is **iNZight**'s foundational design principle and the primary point of difference from other R GUI interfaces. This principle extends to handling specific dataset types, such as those generated by complex surveys: users can interact with, explore, and visualize the data without worrying about the underlying structure.

A fundamental difference between **iNZight** and other R GUIs is its much broader and less technically-skilled audience. We cater to students new to statistics and analysts and researchers in community groups and government agencies with various requirements and skill levels. We use simple design principles to engage our audience and democratise data analysis skills by minimising the demand for up-front knowledge—the names of tests and procedures, for example—and replace it with up-front information easily accessed through contextual dialogues. These principles led to **iNZight** taking on a "variable first" approach. 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. Section 2 includes further details on the design principles of **iNZight**.

iNZight has been adopted throughout New Zealand's statistical education programs, from the year 9 (age 13) school curriculum to postgraduate statistics courses in teaching institutions around the country. In addition, the combination of **iNZight**'s simplicity and powerful toolset make it a popular choice for research organisations and national statistics offices, including the Australian Bureau of Statistics[1] and Statistics NZ.[2] 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 installation process (Section 6).

2. Design principles for iNZight

Democratisation aims to improve the accessibility of data for a broader audience by removing

[1] https://www.abs.gov.au/
[2] https://www.stats.govt.nz/

the barriers restricting access and use, thereby unlocking the ability to discover and apply the information contained within data. These barriers often come down to the availability of time, funding, and the required skills. As we have noted in Wild *et al.* (2021), co-author Sporle is involved with statistical and health agencies in several small Pacific nations, who are often overwhelmed by the information needs of their own countries and the reporting demands of key international agencies such as the United Nations. The national statistics offices in these countries face the triple burden of distance, small workforces, and insufficient funding. They find it challenging to recruit and retain people with good data science skills, and often cannot afford external expertise.

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 require maintaining control over indigenous data resources. Users can access **iNZight**'s data analysis and graphical functionality for free without the need to upload the data beyond their data environment—possession and control over the data are maintained.

The situation is very similar for subject-matter researchers in areas where projects have potential societal value but a lack of 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. There will never be enough altruistic, skilled volunteers to meet these needs. Therefore, it is vital to enable more people to do more for themselves in statistically robust ways. While the R GUIs mentioned in Section 1 do this at some level, some critical 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 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 do various tasks and work in specific processes infrequently. Therefore, working around the rapid fading of memories—such as how to do things and what they mean—is also a significant factor, bringing us to the "problem of names". With programming and most of the existing data analysis GUIs, users cannot do anything until they know what they want to do and remember its name. This problem is a significant barrier to getting started and results in significant time-losses getting back up to speed after a period of inactivity and a subsequent loss of familiarity. Students taking service courses in statistics and data science, particularly those majoring in other subjects, are not immune to the problem either. The long time delays experienced before applying almost anything they learn in class makes ease of refamiliarisation critical.

To circumvent the difficulties described above and seriously empower our users, 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 (DPs):

1. approach tasks in a top-down way with very high-level user requests;

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

2. require as little user input as possible through automation and defaults;

3. use context to display relevant options based on user choices to reduce dependence on up-front knowledge;

4. guide users through complicated procedures by asking for one piece of information at a time;

5. display alternative options only when applicable to the data type or task.

The basic mode of **iNZight** provides visualizations and analyses for rectangular data where variables are in columns and observations in rows. The "variable-first" design lets users drive the software by 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 actually *"Show and tell me about . . . "* and refers to instant graphics delivery whenever users interact with the interface since we believe graphics are the most accessible, information-rich artefacts for broad audiences. Additionally, people are less likely to do silly things after looking at their data first.

Users can obtain numeric information (the "tell" in "Show and tell") by explicitly asking 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"*. Get Inference gives users an analysis of variance, chi-square test, regression panel, or other available test based on the variable(s) selected. Confidence intervals and p-values accompany inferential output wherever appropriate for the tests performed. As per DP 3, users do not have to know or remember what to ask for or how to ask for it. This behaviour demonstrates one of many barriers knocked down by **iNZight**—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. When used in its basic mode, the up-front knowledge requirements of **iNZight** are minimal: users only need a high-level familiarity with rectangular data and variables. The ability to identify situations where a variable-type default may need overriding (for example, when numeric codes are used as group labels) may also be helpful.

In the basic mode described above, we have automated everything using defaults and delivering immediate results. However, the use of defaults begs the question, *"How else can I look at this?"*. As an example, 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?"* (what other methods exist), for example, by making suitable alternatives available (through dropdowns or other controls) in place of the defaults provided. Inferences, for example, can be based on normal theory or bootstrapping, and a switch can turn on epidemiological versions of outputs when appropriate (for example, odds and risk ratios). Options for plot enhancements are extensive, including:

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

- 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 users might desire for aesthetic reasons.

Another feature of **iNZight** is its code-writing functionality, which it shares with other GUIs, but less prominently. Most users will not need to see any R code, but for those who do, all actions within **iNZight** run R code, which gets saved in an R script. In several places, live-code is displayed in an editable box for interactive use (more details in Section 3.6). Initially, 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. Users use **iNZight** as usual but can obtain the R code used to perform the analysis they have just done. This way, R learners can begin editing and running R code themselves without starting from a blank script. This same feature also makes **iNZight** a powerful research development tool. Researchers can quickly explore a dataset and then generate an R script to form the basis of a robust, reproducible workflow in a research setting.

iNZight's high-level, variable-driven requests accompanied by instant results facilitate and encourage rapid data exploration. By significantly reducing the barriers to access, data analysis becomes a skill available to a more diverse range of individuals and organisations. In doing so, **iNZight** is facilitating the democratisation of data, a precious resource in our modern digital world.

3. A tour of iNZight's features

iNZight's ease-of-use has enabled us to broaden our audience from high school students to tertiary institutions, community groups, government, and other researchers. We achieved an intuitive and familiar interface using standard GUI controls such as *drag-and-drop*, *drow-down* selection, and *slider bars*. Users can select variables to explore and drive the software, which reacts instantly to each action using variable type and intelligent defaults to choose the best output. The simplest way to explore **iNZight**'s functionality and illustrate the concepts introduced in Section 2 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). Some of these packages are dedicated to importing the common (and indeed many uncommon) file formats. Usually, R users need to recognise or look up the file extension, find an appropriate package, then decipher the documentation to import an unfamiliar file. **iNZight**

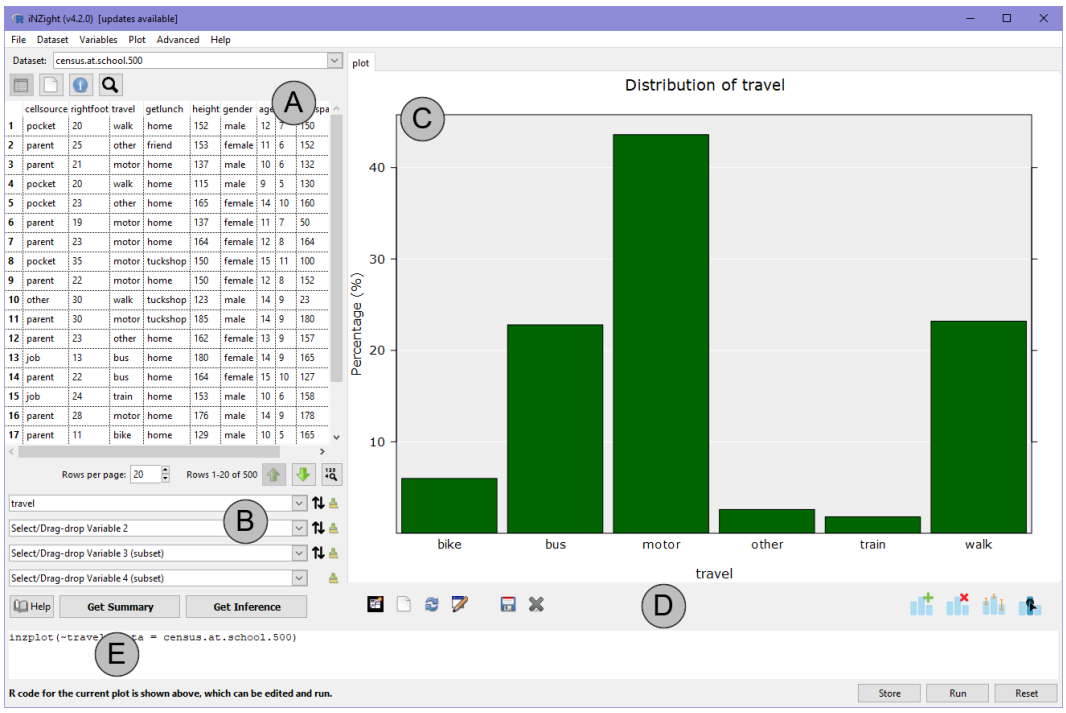

Figure 1: The **iNZight** GUI landing page presents users with a few controls. The labeled areas are: (A) data sets are displayed here prominently, and users can use the controls to switch between data and variable views; (B) variable control boxes provide users either drag-and-drop from (A), or select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

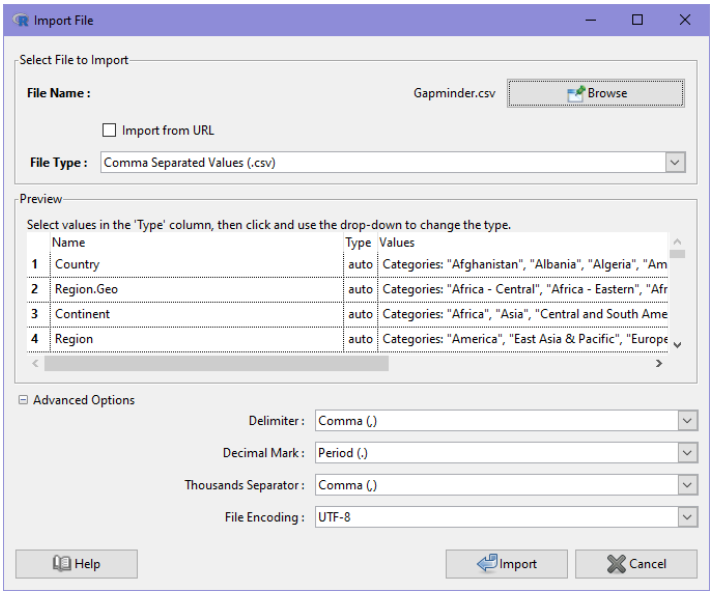

Figure 2: The "Load Data" window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

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

At the time of writing, **iNZight** (version 4.2) 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 to check before proceeding with the import. Figure 2 shows the Import Data window for a CSV file. The Import Data window has an Advanced Options section for CSV and delimited-text files, expanded in Figure 2. Here, users can override the default delimiter or choose between different encoding formats. For example, it is common in European countries to see the semi-colon (;) used as the delimiter, so this option allows users to read files coded this way. The preview is updated when these options are changed, so users can use trial-and-error if they are unsure what the necessary inputs are. This feature is handy for guessing encoding, which can be tricky to work out manually. The data preview also allows users to override the default variable types, which is beneficial when importing a dataset with coded factors. For example, for values $1, 2, 3$ instead of "A", "B", "C", users can override on import the variable type from 'numeric' to 'categorical' and relabel the levels after import.

iNZight also supports copying and pasting data from a spreadsheet to make things even easier for beginners. Data can be copied from Excel or a similar program or directly from a webpage if the formatting is appropriate. Users then open the Paste data from window, which has a text box for pasting the data. As before, a preview displays the result for quick confirmation before proceeding with the import.

3.2. Visualising data with graphics

Since the initial prototype using drag-and-drop to select variables and create graphs, graphics have been at the core of **iNZight**'s user experience. 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, it takes time for beginners to 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 automation, together with the ability to scroll quickly through a set of applicable alternative graphs, also means experienced users can very quickly look at, or scroll through, various 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. Now a second variable can be chosen. Suppose this second variable is categorical, for example, 'ethnicity'. In that case, the software will draw a set of dot plots of height, one for each ethnicity, stacked vertically. However, if *Variable 2* is *numeric*, such as 'age', we are shown a scatter plot of height versus age: *Variable 1* (height) becomes the y-variable in the

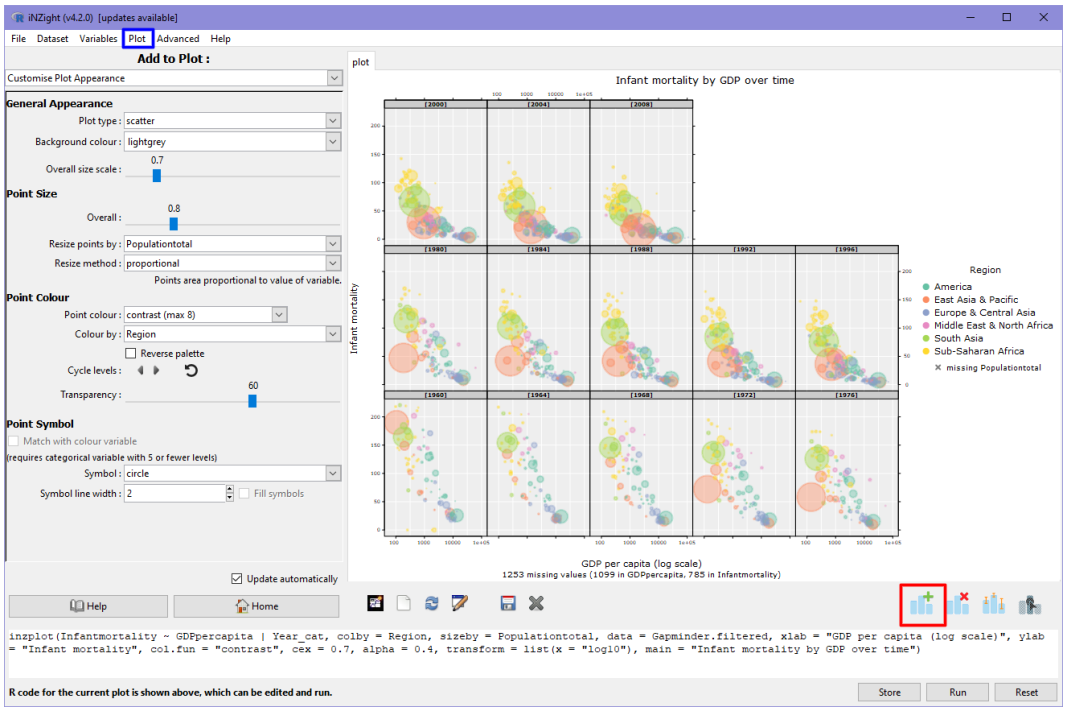

Figure 3: Demonstration of plot modifications available from **iNZight**'s Add to Plot menu. The Add to Plot button, highlighted in red, opens a panel giving user control over colours, size, shape, labels, and much more. This can also be accessed from the plot menu, boxed in blue.

scatter plot. The final 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.

The Add to Plot module is dedicated to plot modifications for a deeper exploration of variable relationships and is accessed from either the Plot menu or the button in the Plot Toolbar (boxed in red in Figure 3). Here, users can choose from a selection of alternative plot types based on the selected variable(s). Other modifications include specifying a colour or sizing variable, adjusting plot symbols, adding trend lines, modifying 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 exploring the data and are not limited by a lack of knowledge or coding skill. Meanwhile, 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 using the *Gapminder* dataset.[5] Here, plot modifications are used to show region (colour), population (point size), and year (subsetting/faceting).

3.3. Numerical summaries and inferences

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

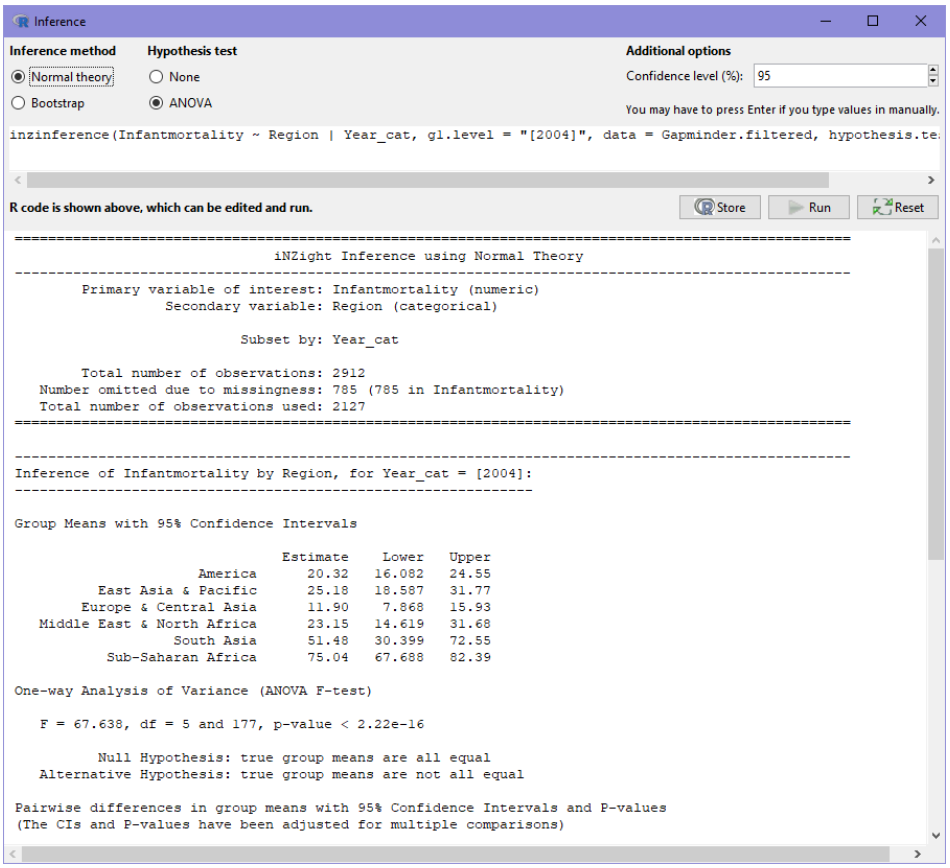

Figure 4: The INFERENCE window provides a selection of hypothesis tests for the chosen variables. In this case, these are `Infantmortality` (a numeric variable) and `Region` (categorical with six levels), so **iNZight** provides an ANOVA test.

Basic **iNZight** provides two textual output modes to supplement the graphical display: *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"*. The summary information includes basic statistics about the variable(s) in the graph: dot plots are summarized by means, standard deviations, and quantiles; bar chart summaries display tables 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.

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. Table 1 gives a complete list of available tests in **iNZight**'s basic mode (as of version 4.2). Users may choose between Normal theory or Bootstrap methods (using the **boot** package, Canty and Ripley 2020) to calculate inference information. We also have plans of adding Bayesian inference methods in future.

Table 1: iNZight hypothesis test options available for various variable type combinations.

Variable 1	**Variable 2**	**Test**
Numeric	(none)	One sample t-test
	Numeric	Linear regression
	Categorical (2 levels)	Two sample t-test / ANOVA
	Categorical (3+ levels)	ANOVA
Categorical		
Two levels	(none)	Single proportion
	Numeric	Two-sample t-test / ANOVA
	Categorical	Chi-square test for equal distributions
Three or more levels	(none)	Chi-square test fir equal proptions
	Numeric	ANOVA
	Categorical	Chi-square test for equal distributions

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 if it is not formatted correctly. Data transformations may be required before any worthwhile 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 whole dataset or 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 filter by value, convert from *wide*- to *long*-form, or join two datasets together. In most cases, the interface evolves from top to bottom as the user inputs information. For example, most processes begin by asking the user to select a variable with which to work. Subsequent inputs are then tailored to previous choices, thus guiding users through the procedure (DP 5). At the bottom of many data manipulation windows is a preview of the data after transformation, demonstrated in Figure 5, once again allowing users to play with inputs to get the desired output before committing to a transformation. Appendix A.1 contains a complete list of available methods.

The Variable menu gives users access to a range of variable transformation and modification actions. For example, users can convert numeric variables into categorical ones or modify the levels of a categorical variable by renaming, reordering, or combining them. Users can also create new variables based on others in the dataset and rename or delete existing variables. In all cases, **iNZight** creates a *new* variable for each action. For example, converting `Year` to categorical yields a new variable named `Year_cat` by default. This behaviour makes the experience more transparent and exploration-friendly. Appendix A.2 gives a list of available variable manipulation methods.

3.5. Additional facilities for special data types

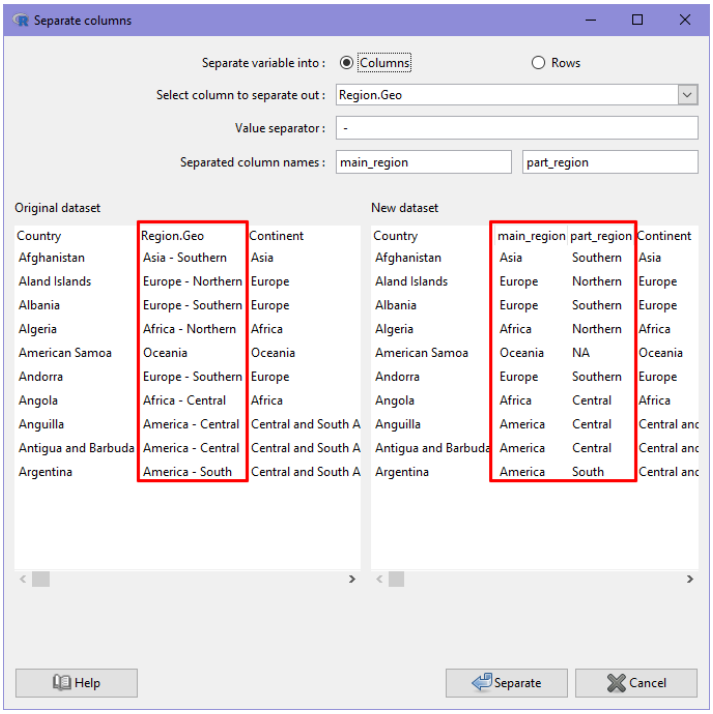

Figure 5: Here, the user is separating a column to create two new variables, with the preview displayed in the bottom-right. The relevant columns are boxed in red. The preview uses the first few rows of the data, and updates in real-time, reacting to changes the user makes, allowing them to experiment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

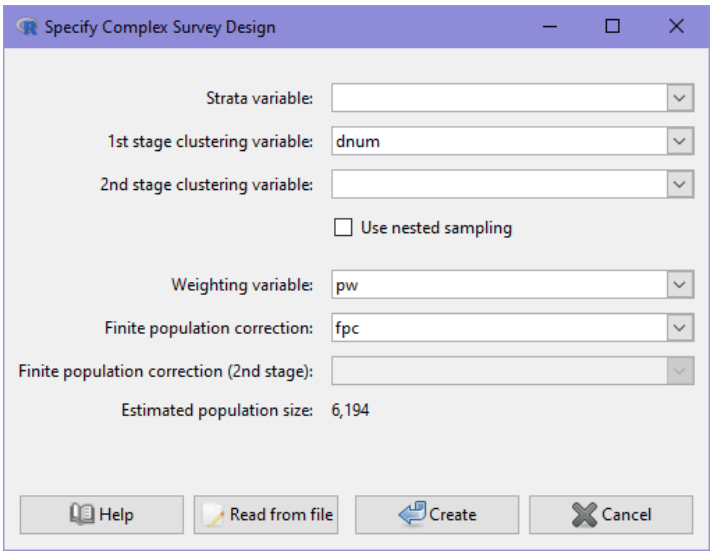

Figure 6: Users can specify survey design information manually by filling in the fields. These will then be used throughout the session thereafter where appropriate.

Beginners are often exposed to data sets that are in 'tidy' format (Wickham and Grolemund 2017, chapter 12), such that rows represent individual records and columns contain variables. There are, of course, several types of data that are a vital component of statistical analysis or are commonly encountered by those new to the field. Such datasets require specialist graphics or handling, often 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 essential data types demographers and researchers who use official statistics encounter is data from complex surveys, which require information about the survey's structure to provide valid graphs, summaries, and inferences. **iNZight** handles survey designs behind the scenes, requiring the user to specify the structure manually (Figure 6) or import a *survey design specification* file. Data producers can distribute the design file with their datasets to ensure that users conduct valid data analyses. Once specified to **iNZight**, the user can forget about the survey design and use **iNZight** as usual. 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. **iNZight**'s code history automatically includes the R code (Section 3.6) for specifying the survey design.

iNZight handles data from complex survey designs involving staratification, one- and two-stage clustering, and unequal probabilities of selection. Additionally, **iNZight** supports *replicate weight designs* where a set of replicate weights replaces design information such as clustering and stratification variables. These replicate weights allow surveys to be analysed correctly without exposing confidential or private information 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

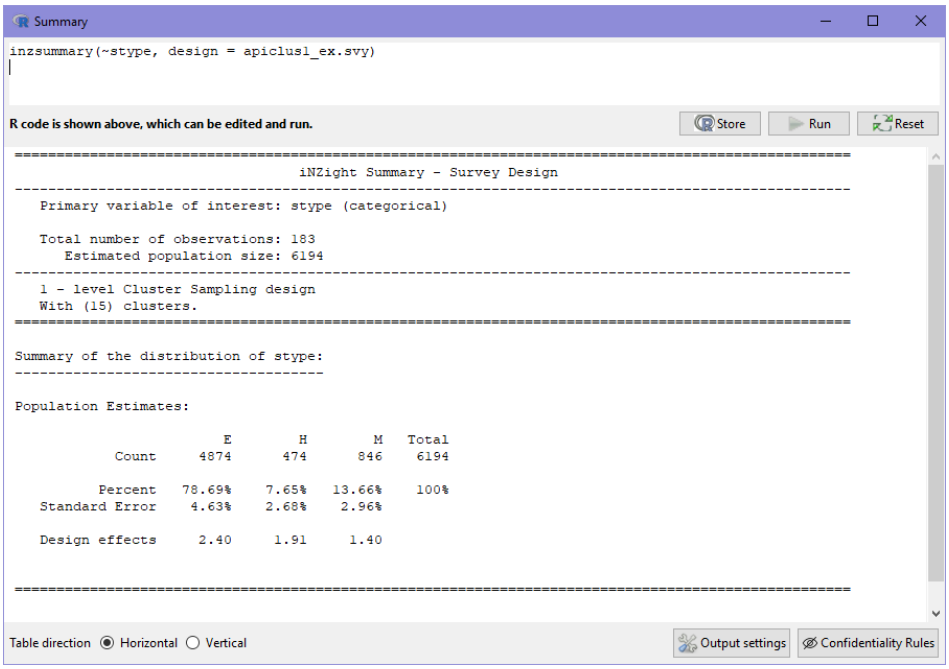

Figure 7: The SUMMARY window provides simple summary statistics and, in the case of survey data, standard errors of these population estimates.

data from other sources to reduce the variances of estimates. Once again, this information is specified once (possibly using a design specification file) and used throughout the analysis.

The types of graphs available differ for survey data, as they now represent a picture of the *population*, not simply a sample. For a single numeric variable, a *histogram* is displayed by default instead of a dot plot. A *bubble plot* is used for two numeric variables, which is a scatter plot with points sized by the weights of respective observations. Alternatively, a *hex-bin* plot can be used to visualize the relationship between two numeric variables and is particularly effective when there are large numbers of observations; indeed, it is the default in such situations. 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 appropriate methods from the **survey** package.

Time series

Another standard data type is *time series*, in which the variable of interest is observed changing over time. Time series data can be explored with the dedicated *Time Series* module in **iNZight**. Time information can be specified either in a specially formatted variable or manually by the user. At the time of writing, **iNZight** only supports time series with equally spaced observations and non-missing values. However, uneven observations can be aggregated from the VARIABLE menu to prepare the data for the time series module.

iNZight's time series module allows users to plot one or more time series on a graph to see or compare how values change over time. The software automatically overlays a smoother

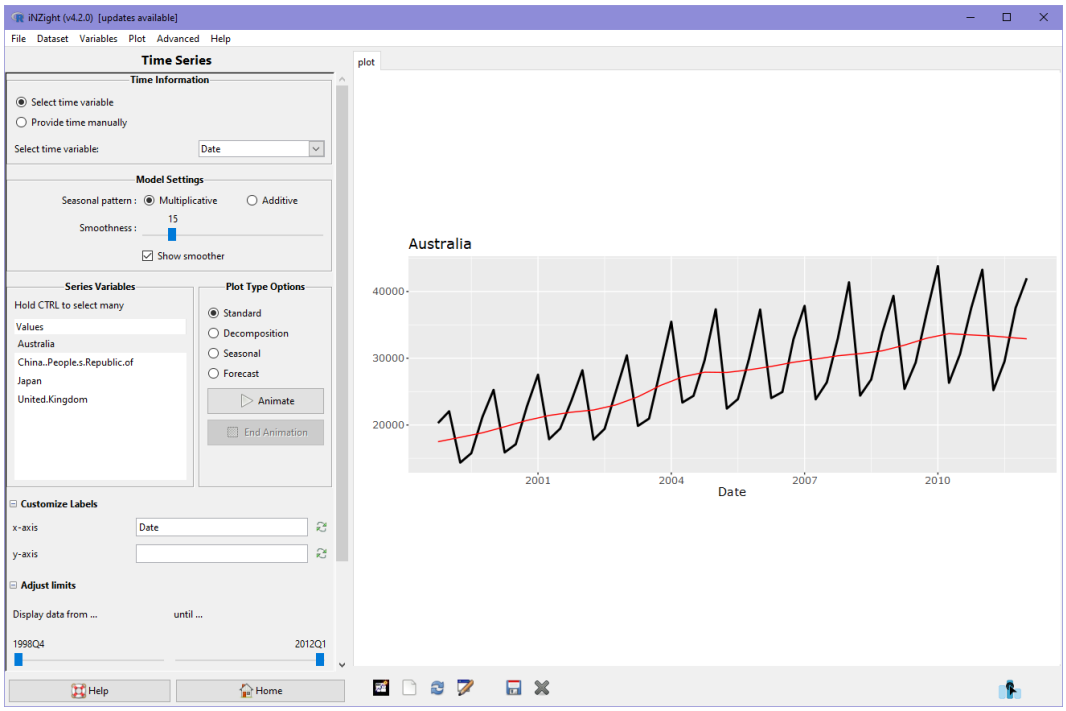

Figure 8: The time series module. This data set contains a variable called 'Date' which is in the correct time format, so **iNZight** automatically plots the first variable, in this case 'Australia'.

across each series. Additionally, the series can be decomposed into the trend, seasonal, and residual components using seasonal-trend decomposition using LOESS. Optional animations help users understand how the various components combine to form the final series, and the *Forecast* plot type provides a Holt-Winters' forecast (Holt 2004; Winters 1960).

Figure 8 shows the time series module with quarterly visitor arrivals data for several countries. When loaded, the software automatically detects the `Time` column ("Date") and draws the displayed graph without any user interaction. Users can choose between *additive* and *multiplicative* models and use the slider to control the smoothness of the LOESS curve (in red). Users choose one or more variables from the SELECT VARIABLE list to display on the graph, while graph type is selected from 'Plot Type Options'.

We currently have work underway transforming the time series module from base R to **tidyverse** using **tsibble** and related packages (Wang *et al.* 2020). This will enable working with dates in a range of formats, enable automatic interpolation of missing values, and provide a more diverse range of forecast methods.

Maps

Geographical data is essential for looking at regional effects or the distribution of location-based events; however, it can be difficult to create appropriate graphs, often requiring maps or shapefiles. **iNZight** features a *Maps* module for exploring two types of geographical data: point-based data, in which observations have latitude and longitude locations (for example, earthquakes), and regional maps for exploring data related to fixed regions (for example,

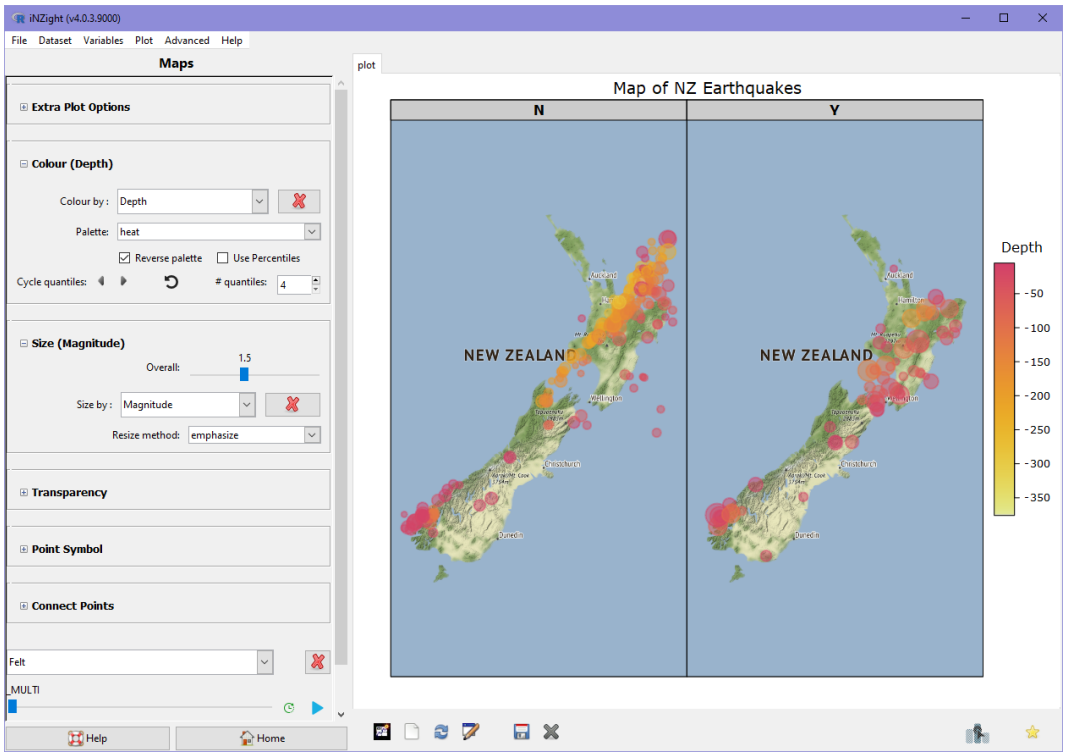

Figure 9: The maps module showing New Zealand earthquakes sized by magnitude, coloured by depth, and subset by whether or not they were felt.

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

Point-based observations use the **ggmap** package (Kahle and Wickham 2013) to overlay a map with observed data. The maps module in **iNZight** lets users explore other variables in the dataset using the same techniques used for scatter plots: users adjust the size, colour, and transparency of points and specify faceting using an interface similar to the base program. A demonstration of this module using New Zealand earthquake data[6] is shown in Figure 9, 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 (Auckland or Otago, for example), with physical boundaries that an external shapefile can describe. Another example is the boundaries of the states of the USA. 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 visualise on the map using colour, points, and other techniques. Users are often interested in regional trends over time, so **iNZight** can automatically detect longitudinal or time series data and animate maps over time or display trends as *sparklines* (Tufte 2001).

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

Other data types and features

Besides these examples, **iNZight** has several other modules that allow more complex statistical methodology or support other unique data types. *Multiple Response* data arises from "*Choose all that apply*" type survey questions and need their own graphics methods to explore adequately. The multivariate data add-on module lets users perform multivariate statistical methods and visualise the results, including principal components analysis and non-metric multidimensional scaling. Using the model fitting module, users can fit complex linear and generalised linear regression models to standard and survey data. In this module, regression output automatically updates as users add and remove explanatory variables. A range of residual plots is available to explore and help users 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 data types or 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

One feature prominent in the other R GUIs is the coding interface, which differs significantly from **iNZight**'s. For example, R Commander provides a prominent "script" box into which code appears when using the command boxes, or users can enter custom 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. Also, each of the GUIs requires familiarity with R coding and an understanding of simple statistical terms and methods to navigate the menus. In contrast, **iNZight** runs separately 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, including importing the data, applying transformations and manipulations, and any plots and summaries the user chose to save. The script provides a record of what the user did and can be saved, edited, and run in R. This lets users explore a dataset with a GUI tool to quickly start an analysis and generate an R script template to use as the basis of a reproducible workflow.

A more advanced feature is the interactive R code box at the bottom of the interface (see Figure 1). It displays the code used to generate the current plot, and, more importantly, users can edit and run the code displayed. 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 so that it is added 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 as 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, easy-to-use, high-level functions in the **iNZightPlots** package (Elliott *et al.* 2020) 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 built with R using the three support packages **gWidgets2**, **gWidgets2RGtk2**, and **RGtk2**. **gWidgets2** (Verzani 2019) provides a simple widget-based application programming interface (API) to build 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 **GTK+ 2.0**, as it was the most feature-rich and—at the time—had the best cross-platform support (see Section 6.1). The **GTK+ 2.0** binaries are accessed through R using the **RGtk2** package (Lawrence and Temple Lang 2010), with commands translated from **gWidgets2** using the **gWidgest2RGtk2** package (Verzani 2020).

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). **gWidgets2** uses the same framework to describe individual components. Each piece of the GUI is a *class*, with individual buttons, methods (actions), and even smaller sub-components. OOP features *inheritance*, allowing developers to create a general class shared by several related 'child' components, but with different layouts or methods; this drives **iNZight**'s add-on system (Section 5). Figure 10 shows the **iNZight** GUI with several main components annotated.

In addition to the "visible" components, others exist behind the scenes. The main one is the '`iNZDocument`' class, which stores the application's state, including the data set, variable selection, 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. This is invaluable when performing data transformations such as aggregation and reshaping, as each action creates a new document.

The structure of each component is, in most cases, a set of attributes that the user can control, stored as *properties*, along with *methods* that the component can use to react to user input or perform actions. Most components have a main method that performs the primary function of the component. For example, the '`iNZFilterData`' class contains a `filter_data()` method that takes the user's input and performs the desired action. Listing 1 provides a simplified example of the Filter Data window class, providing the user with a drop-down `gcombobox()` to choose a variable to filter the data. When they click the Filter button, the data is filtered and passed back to the main GUI. The method uses the `switch()` function to select the appropriate wrapper function within the **iNZightTools** package based on the user's chosen value of "type". The actual class for the Filter Data method is more complicated and includes reactive components, so only the relevant inputs are displayed to the user.

Each major component's structure is similar 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 sep-

```
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.

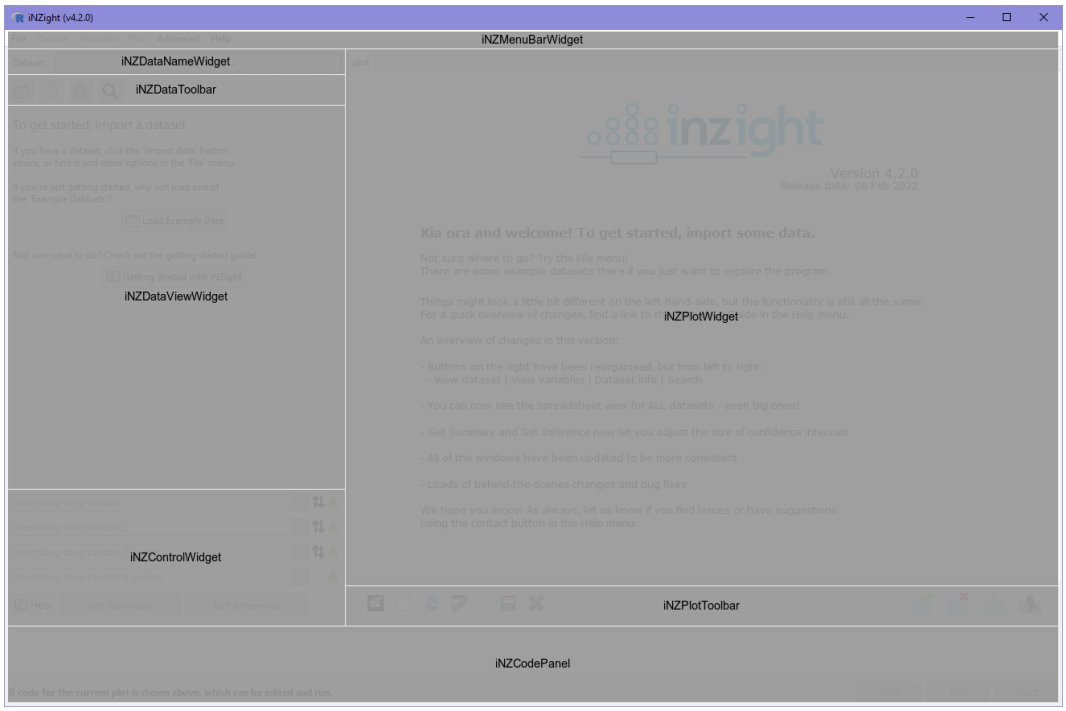

Figure 10: The reference class components of the **iNZight** interface, some of which are themselves made from several child objects.

aration 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. The GUI can fetch the code from the returned data and attach it to the script (described in Section 3.6) while keeping the GUI and data-oriented code separate. Here is an example using `iNZightTools::smart_read()`:

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

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

The `iNZightTools::code()` function can extract the R code attached to the resulting object. In the example above, R users can 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 quickly examine and modify the underlying code returned. In this way, these users can read a wide variety of datasets and 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.

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.

While the GUI packages provide the structure of the visual GUI, it is the collection of R packages developed alongside **iNZight** that power the program. The main reason for creating separate packages was to separate interface and data logic. Additionally, it allows parallel development of our Shiny (Chang *et al.* 2021) version (Section 6.4) using the same wrapper functions. Table 2 lists the packages associated with the **iNZight** project. Most of these packages have simple high-level interfaces that connect to the GUI but can also be used by beginner R users on their own (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 is launched by a function of the same name:

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

This function takes an optional `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)
```

The GUI object is returned invisibly for development purposes and may be assigned to a variable:

```
ui <- iNZight()
```

This way, developers can access the '`iNZGUI`' object and can explore states and trigger actions for easier testing. In the following 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

The main **iNZight** program will have all most users need to explore, visualise, and perform simple analyses on their data. Some users, 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 **iNZight** modules that can connect to new or existing R packages available on CRAN or elsewhere.

Installing existing add-ons is easy. Users can add, update, and remove modules from our add-on repository,[7] a custom URL, or a local file from the MODULE MANAGER. 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**. When opened, the module has access to the **iNZight** interface, including the dataset imported by the user.

The module files themselves describe a single class object inherited from '`CustomModule`'. This parent class provides several methods, including initialising the module panel on the left-hand side of the **iNZight** interface. Developers can write additional properties and methods for individual modules, opening up possibilities for teachers, research groups, or even R package developers themselves to write custom modules to distribute to their audiences.

Figure 11 demonstrates a prototype of 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 process involving non-linear transformations of estimated mortality rates. Bayesian methods, however, are easy, although they often involve much coding to implement. The **iNZight** small area estimation module Elliott and Bryant (2021) provides a simple interface to several R packages that implement Bayesian demographic estimation: **dembase** (Bryant 2020), **demest** (Bryant *et al.* 2020), and **demlife** (Bryant and Peterssen 2021). However, these packages are currently under development, so a full version of the **iNZight** module will be released once they are complete.

6. Installation and availability

iNZight can be installed in R from our package 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

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

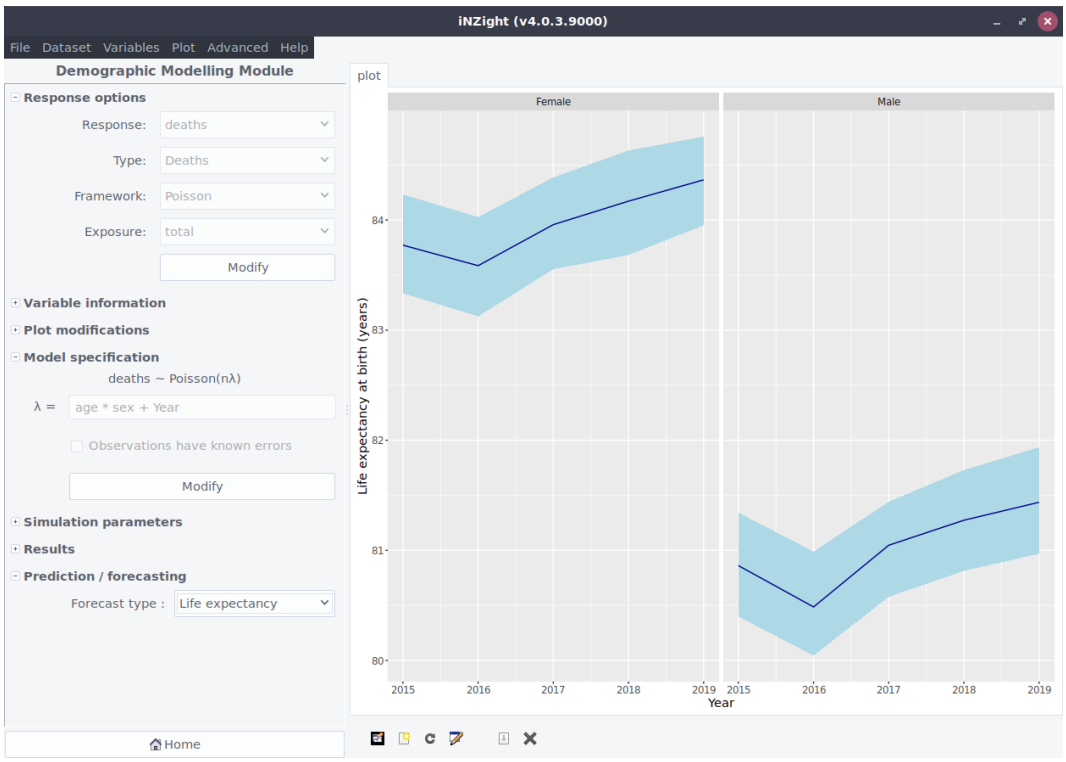

Figure 11: The prototype for a new Bayesian demographic modelling module for **iNZight**. In this example, life expectancy is estimated from death data.

iNZight is a GUI, additional system dependencies 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 Windows, macOS, and Linux libraries. However, the install process varies between operating systems in both steps and complexity. On Windows, the necessary files are available as binaries 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 must 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`, 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. Several other system dependencies need installing for some features of **iNZight**. For the latest list, check `inzight.nz/install`. Linux users can install **iNZight** by running the Windows installer (Section 6.2) under Wine.[8]

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 that includes a copy of R, the package library, and GTK, so once installed, **iNZight** is ready to go. By default, **iNZight** is installed into the user's `Documents\iNZightVIT` directory.

In addition to the binaries and packages, the installer includes several Desktop shortcuts that launch R in a specific directory. This directory contains a `.Rprofile` file that automatically loads the **iNZight** package and launches the interface. It also hides the R console, presenting users with just the GUI. R receives a command to terminate R once the user has finished using **iNZight** when started from this script.

The **iNZight** installer also includes an *Update* script to allow easy updating of the R packages. Novice users can update to the latest version without using R or downloading the latest installer. Additionally, we include an Uninstaller that removes **iNZight** from the user's system by deleting the folder and any shortcuts.

6.3. Docker image

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

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 that end users can download to run the program without installing numerous 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 is available 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 can run R and, therefore, **iNZight**. To cater for such devices, we developed an online version of **iNZight** called **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. Further, 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 a docker container on a remote Amazon Web Services server.[9] 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. A set of URL parameters allows configuration of the **iNZight** *Lite* instance, including a URL for a dataset to load automatically when the user connects. For example, it is possible to store datasets on a server linked to URLs to launch *Lite* 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 in action is available at `https://www.stat.auckland.ac.nz/~wild/data/Rdatasets/`.

The **shiny** package is used to create visual controls and perform reactivity events. A user's data is temporarily stored on the server and is only accessible from that 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 better be explored using either the desktop version or by running **iNZight** *Lite* locally. Research groups could host a build of *Lite* with access to private data.

7. Summary and future work

TODO: less student, more national stats offices ...

Newcomers to statistics often need to learn how to code using R whilst simultaneously learning the basic skills for data exploration. In contrast, many researchers need quick, easy tools to get new projects started. By providing an easy-to-use GUI, **iNZight** lets users focus on exploring

[9] https://aws.amazon.com/

and analysing data. Beginners can develop interpretation skills before embarking on the more challenging part of learning to code. Using a *variable first* approach means 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), guiding them through the analysis.

Similarly, data manipulation techniques like filtering, renaming factor levels, and specifying survey designs are presented in simple step-by-step windows, many of which provide previews. This way, users can tweak the input controls to get their desired output. **iNZight** also includes simple tools for users learning to code with R. All code is written to an R script for users to review. Alternatively, the reactive code panel lets users interact with code without leaving **iNZight**.

Statistics and data science are ever-expanding fields, 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. In so doing, we are democratising critical methods and skills, making them available to a broader range of communities and organisations worldwide.

7.1. Future Work

- further developing iNZight for surveys, particularly social research, and extending to native handling of multi-part variables (e.g., those from multiple response questions)
- redevelopment of the time series module to make use of recent advances in packages such as 'tsibble', 'feasts', and 'fable', which will allow extension of the module to a wider range of time series data
- connecting to data bases, and specifically allowing on-the-fly queries—particularly useful for joins (e.g., in large linked datasets) where trying to have all datasets/variables in memory would be unfeasible
- using 'specification' files for surveys and collections of datasets so complex data structures can be described to the software in a simple way, allowing users to one-click load surveys/linked or longitudinal data sets (for example) and use the software as usual.
-

Many new features and functionalities are planned for **iNZight**. The foremost is interacting with more complex datasets, particularly those saved within a database. Analysis of more data types, particularly surveys and longitudinal data, is actively being explored.

The main issue with **iNZight** is its reliance on GTK, which is discontinued on macOS. We are exploring alternative frameworks for **iNZight** to create a fully cross-platform application and remove even more barriers restricting access to 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.

Bryant J (2020). *dembase: Analysing Cross-Classified Data about Populations*. R package version 0.0.0.119.

Bryant J, Harlow J, Zhang JL, Taglioni C, Wang F (2020). *demest: Bayesian Demographic Estimation and Forecasting*. R package version 0.0.0.3.1.

Bryant J, Peterssen K (2021). *demlife: Life Tables*. R package version 0.0.0.9007.

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, Bryant J (2021). "Bayesian Demography with iNZight." *Technical report*, Te Rourou Tātaritanga, Victoria University of Wellington. URL https://terourou.org/outputs/inzight_demography/.

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/.

Tufte ER (2001). *The Visual Display of Quantitative Information*. 2nd edition. Graphics Press.

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.

Wang E, Cook D, Hyndman RJ (2020). "A new tidy data structure to support exploration and modeling of temporal data." *Journal of Computational and Graphical Statistics*, **29**(3), 466–478. doi:10.1080/10618600.2019.1695624. URL https://doi.org/10.1080/10618600.2019.1695624.

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

Menu item	Description	Example tidyverse code
Filter	Filters dataset by levels of a categorical variable, value of a numeric variable, specific rows, or randomly.	`dplyr::filter(data, variable > x)`
Sort by variable(s)	Sort dataset row (order) according to one or more variables.	`dplyr::arrange(data, variable)`
Aggregate	Compute summaries of selected values grouped by one or more categorical variables.	`dplyr::group_by(data, cat_var) %>% dplyr::summarize(...)`
Stack	Stack two or more columns into a single column with a new 'key' column. This is a simple version of reshaping from long to wide form.	`tidyr::pivot_longer(data, vars)`
Reshape dataset	This provides a more advanced interface for converting between long and wide formats.	`tidyr::pivot_longer(data, vars)` `tidyr::pivot_wider(data, names_from = key, values_from = value)`
Separate column	Split a column in two at a specified character.	`tidyr::separate(data, var, c("a", "b"))`
Unite columns	The reverse of 'split', allows two columns to be joined together with a separator to create a new variable.	`tidyr::unite(data, "new_var", c("a", "b"))`
Reorder and select variables	Allows users to select a subset of variables, and order them.	`dplyr::select(data, ...)`
Join by column values	Perform inner/outer joins of two datasets.	`dplyr::inner_join(data1, data2)`
Append new rows	Allow joining of two related datasets (i.e., with the same column names), for example if you have separate files for different years of data.	`dplyr::bind_rows(data1, data2)`

A.2. Variables

Capabilities to modify, create, and delete existing variables in a dataset.

Menu item	Description	Example tidyverse code
Convert to categorical*	Convert a numeric variable into a categorical one.	`dplyr::mutate(data, x_cat = as.factor(x))`
Categorical Variables		
Reorder levels	Reorder the levels in a factor/categorical variables.	`dplyr::mutate(data, x_reord = factor(x, levels = c(...)))`
Collapse levels	Collapse multiple levels into one.	`dplyr::mutate(data, x_coll = forcats::fct_collapse(x, ...))`
Rename levels	Rename levels of a categorical variable	`dplyr::mutate(data, x_rename = forcats::fct_recode(x, ...))`
Combine categorical variables	Create a new categorical variable that is the combination (or cross) of two or more variables.	`dplyr::mutate(data, x_y = forcats::fct_cross(x, y))`
Numeric Variables		
Transform	Apply one of several common transformations (e.g., log, square-root) to a numeric variabel.	`dplyr::mutate(log_x = log(x))`
Standardise	Standardise a numeric variable to have mean 0 and standard deviation 1.	`dplyr::mutate(std_x = scale(x)[, 1])`
Form class intervals	Convert a numeric variable into a categorical one by creating class intervals of the form $[a, b)$, $[b, c)$, etc.	`dplyr::mutate(x_f = cut(x, ...))`
Rank numeric variables	Create a new variable indicating the rank (order) of values in the chosen variable(s) from lowest to highest.	`dplyr::mutate(x_rank = dplyr::min_rank(x))`
Convert to categorical (multiple)	Same as * but allows users to convert multiple variables at once.	
Dates and Times		
Convert to	Convert a variable into a date/time	`dplyr::mutate(x_dt = lubridate::parse_date_time(x, ...))`
Extract from	Extract a specific part of a date/time variable (e.g., just the month, or year)	`dplyr::mutate(x_dt_month = lubridate::month(x_dt))`
Aggregate to	Aggregate data into wider intervals by taking the mean/median/min/max/sum of values in each period (e.g., convert between total visitors per month to per quarter).	`dplyr::mutate(data, quarter = lubridate::quarter(x))` `dplyr::group_by(quarter)` `dplyr::summarise(...)`
Rename variables	Rename variables in the dataset	`dplyr::rename(data, ...)`
Create new variables	Allows creation of (arbitrary) new variables, e.g., by taking the different or ratio of existing ones.	`dplyr::mutate(data, bmi = weight / height^2)`
Missing to categorical	For numeric variables, a new variable categorical variable is created with values 'Missing' and 'Not missing'. For categorical variables, NAs are converted to a new level 'Missing'.	`dplyr::mutate(data, x_miss = forcats::fct_explicit_na(x, na_level = "missing"))`
Delete variables	Allows deletion of variables	`dplyr::select(data, -x, -y)`

B. Code history

The R code history generated during the demonstration in Section 3 is copied here. Note that the gapminder data can be downloaded from https://inzight.nz/data.

```
# iNZight Code History

## This script was automatically generated by iNZight v4.2.0

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

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

# install.packages(c('iNZightPlots',
#                    'magrittr'),
#     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("Gapminder.csv",
    comment = "#",
    col_types = readr::cols(),
    locale = readr::locale(
        encoding = "UTF-8",
        decimal_mark = ".",
        grouping_mark = ","
    ),
    lazy = FALSE
) %>%
    dplyr::mutate_at(
        c(
            "Country",
            "Region-Geo",
            "Continent",
            "Region",
            "Year_cat"
        ),
        as.factor
    ) %>%
    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",
    cex = 0.7,
    alpha = 0.4,
    transform = list(x = "log10"),
    main = "Infant mortality by GDP over time"
)

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

## Load example data set
data(apiclus1, 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)

## 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