iNZight: a GUI for Learning Statistics

Tom Elliott and Chris Wild and Daniel Barnett and Andrew Sporle

February 2, 2021

Abstract

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

1 Introduction

- Scope out the need for iNZight

- R can be daunting for beginners/students (who may never have used or even see code before)
- Excel (Microsoft Corporation, 2018), SPSS/etc are rather complex with a not-insignificant learning curve to be able to produce basic exploratative plots and summary statistics
  - Do not provide any kind of a ‘pathway’ to learning R for data science purposes, either
- Other R-based GUIs:
  - Rcommander (Fox, 2005)
  - Jamovi (The jamovi project, 2020)
  - ???
- Other tools targeting students:
  - NZGrapher (Wills, 2020) uses PHP
- No obvious—simple—point-and-click interfaces for simple data analysis/visualisation that also provide a pathway to more complex, code-driven analyses

2 A history of iNZight

- Originally a simple implementation experimenting with R (R Core Team, 2020b) and ‘gWidgets’ (now superceeded by ‘gWidgets2’, (Verzani, 2019)) for making graphs which react to the type of data (i.e., the user doesn’t have to choose the graph type)

- Uses GTK (The GTK+ Team, 2020) to produce graphical interface, accessed via the ‘RGtk2’ R package (Lawrence and Temple Lang, 2010)
- The software uses the variable types (numeric or categorical) to determine the type of graph or summary produced
- Picked up by [. . . ?] and rolled out for use in NCEA Level 3 statistics in New Zealand (final year of high school)
- Redesigned in 2014 with gWidgets2 and reference classes (one of R’s Object Oriented Programming approaches)
- Additionally uses a suite of complimentary R packages to separate form (the UI) from function (data processing, graphics, etc)
- Additionally modules for time series, model fitting, etc, and more recently added an add-on system
- Most of the work has been student-driven: “By students, for students” (rather than being created by computer scientists)

3 An overview of iNZight’s structure

Producing cross-platform GUIs has always been a difficult task as different operating systems (OSs) implement different display devices. Therefore many projects have been created in an attempt to make cross-platform applications a possibility. One such example is GTK+, which is implemented on Windows, macOS, and Linux systems, providing a single toolkit for creating GUIs for all major systems (The GTK+ Team, 2020).

Of course, interfacing with such a framework is in itself a difficult job, and requires some complex C++ coding. Fortunately, several interfacing packages have been written in R (R Core Team, 2020b) which prove a simple, platform (and indeed toolkit) independent application programming interface (API) for writing GUIs from R. The ‘RGtk2’ package (Lawrence and Temple Lang, 2010) provides an platform-independent interface between R and GTK+2, allowing access to most of the classes to construct a GUI that reacts to a user’s input. Additionally, the ‘gWidgets2’ package (Verzani, 2019) provides a framework-independent inteface between R and several other R packages responsible to creating GUIs, namely ‘gWidgets2RGtk2’ for communicating with ‘RGtk2’ (Verzani, 2020). Together, these packages make it possible for any R programming to construct a graphical application without any knowledge of GTK or platform-specific development. Indeed, it is the combination of these tools which made it possible for statistics students to create and work on ‘iNZight’.

Given a platform- and framework-independent API, the next critical step is planning the internal structure of the application, most importantly ensuring that future development will not become hindered by early decisions. The most foundational decision in the early development of ‘iNZight’ was to separate *form* from *function*: that is, the code the controls the interface should be, as far as possible, separate from the code that handles data processing, graphics, and so on. Further, we wanted the individual components of the GUI to be independent to ensure future development would be easier: for example, buttons can be moved and replaced, and new components can be added or old ones removed without affecting anything else. The necessity for object oriented programming (OOP) was clear, so that each individual component is represented by a single *class*, which could be modified independently of others, or modified (“inheritance”) to make similar widgets with common behaviours but several unique features.

Within the R system there are several OOP implementations, however we chose to use *reference classes* as these are part of base R and also used by 'gWidgets2'. Each component of the 'iNZight' interface is defined using one or more reference class objects, each of which has a set of *fields* and *methods*. Fields describe properties associated with the objects, most importantly user-specified values (for example an Import window might have a field for the file name). Methods are functions which carry out actions, and have access to the object's fields (the Import window might have a method to load the data, for example).

To separate form from function, methods that perform actions on the data or generate output for the user call functions in other packages. The interface allows users to specify values for the object's fields, and either automatically or by clicking a button call the methods which compile the fields and pass them to an external function. For example, an Import data window might take the file chosen by the user and pass it to `iNZightTools::smart_read()`, a function in the 'iNZightTools' package (Elliott et al., 2020a) which acts as a wrapper for several other data import functions. In this way, the interface does not need to know anything about the data type it is loading: it only collects information from the user and passes it to another function. An early advantage of this was the creation of 'iNZight Lite', an 'shiny'-based alternative to the desktop version of 'iNZight' which can run in a user's browser. In this case, both of the GUIs collect information from the user—potentially in a different way—and pass it to the external function. This way, the result returned (given the same input) will be the same across implementations of 'iNZight'.

A further advantage of the code-separation is that it provides the opportunity of a "stepping stone" between using the GUI and coding directly in R using specialised packages and functions. Rather than learning all of the different data import functions in a range of packages (such as 'readr' (Wickham et al., 2018),

Table 1: 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 `svyglm` objects.
iNZightTS	Time series visualisation, decomposition, and forecasting.
iNZightMR	Visualisation and estimation of multiple response data.
iNZightTools	A suite of helper functions for data process and variable manipulation.

‘readxl’ (Wickham and Bryan, 2019), or ‘foreign’ (R Core Team, 2020a)) they need only learn one function `smart_read()` to import CSV, Excel, SPSS, Stata, or SAS files.

- examples:
  - `iNZightTools::smart_read()` imports a dataset based on its extension - user doesn’t need to know `read_csv()`, `read_dta()`, etc
  - `iNZightPlots::inplot()` is the main power-house function within iNZight - takes UI inputs and generates a graph based on the variable types (and other selections)
  - both of these functions can be accessed directly from R
  - most also return the ‘tidyverse’ code so learners can get a taste for the actual code necessary to do stuff (e.g., filtering data, etc)

iNZight’s code separation has led to an entire family of R packages, which are displayed in table 1. Most of these packages can be used standalone, and provide simple wrappers for commonly used R workflows, replicating many of the behaviours in *R for Data Science* (Wickham and Grolemund, 2017). Further, most of the functions return the ‘tidyverse’ (Wickham et al., 2019) code used behind-the-scences, providing a further stepping stone for users to learn these more complicated code workflows.

Returning to the underlying structure, we now get a glimpse of the importance of those early decisions on the future development prospects of 'iNZight'. Using reference classes allows us to add and alter individual components without affecting others, and simultaneously providing a singular interface with an external function. Changes to those functions are automatically inherited by 'iNZight', such as storing of the underlying code. Since the data structure is also itself a reference class object, it can apply methods when a change to the data is triggered, namely looking for attached code and appending that to the code history widget.

4 Features of iNZight

At its heart, iNZight is a data visualisation and exploration tool for those users with little to no prior experience with data science or statistics, and who lack the programming demands of more mainstream tools such as Python (Van Rossum and Drake, 2009) and R (R Core Team, 2020b). Therefore, many of the main features relate to exploring data through visualisation, with some data manipulation techniques built in, including specification of survey designs which are automatically incoporated into the rest of iNZight. Since many users will likely want to move on to coding, and since iNZight is built with R, we provide the code history for actions the user makes.

4.1 Data wrangling

The first thing most users will want to do is import their data. iNZight provides an easy to use *Import Data* window which uses the file extension to detect the file type and provide a preview of the data in the same window. This allows users to quickly see if everything is OK and, if necessary, adjust some of the type-specific options to get it correct. An example of this might be reading european CSV files, which use a semi-colon delimiter instead of a comma.

Once loaded, iNZight provides several important data operations, allowing users to reshape, filter, and otherwise transform their dataset. Many of these 'workflows' are taken from "R for Data Science" (Wickham and Grolemund, 2017). These basic dataset operations are implemented using packages from the 'tidyverse' (Wickham et al., 2019). For each, the GUI provides an interface with inputs corresponding to various arguments, generating an R code call which is evaluated and stored in the script. In some cases, a preview of the resulting dataset is provided, making it easier for users to investigate the result of different options. FIGUREX shows the reshape data window, allowing users to convert from wide format to long format, which is more useful for plotting.

- uses 'tidyverse' methods and workflows to perform some data transformation
- all are calls to wrappers inside the 'iNZightTools' package
- the interface allows users to manipulate arguments to the wrapper function - reactive in some cases (i.e., inputs appear/disappear or change values based on previous inputs)
- in many cases, a preview is displayed to help with what can be complex actions, e.g.

As well as the dataset operations are *variable manipulations*, allowing users to modify individual variables in the data. For example, simple transformations (log, square-root) to renaming or reordering levels of a categorical variable, each has its own interface window that interfaces with 'tidyverse' code to perform the operations. And, if the operation you want is not available, you can specify a custom command to create a new variable.

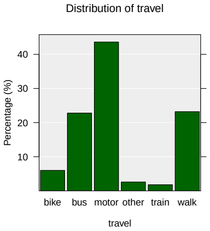
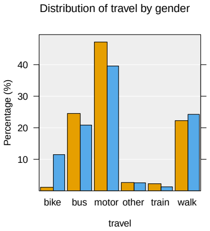
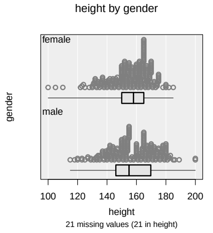
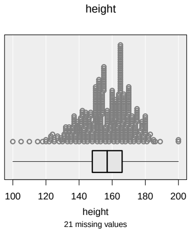
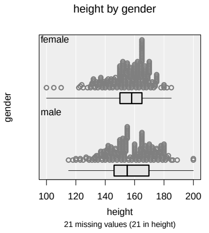
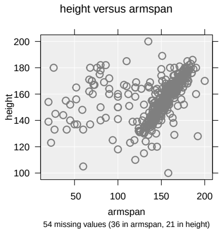
The goal of these features is to allow users to import a range of data sets in a range of formats and convert them into a form useful for plotting—that is, *tidy format* (Wickham and Grolemund, 2017), where each row contains a single set of observations about an individual.

4.2 Graphics and simple data analyses

The foremost tool in iNZight's inventory is graphics, which are chosen automatically based on the users's chosen variables. For example, a numeric variable is displayed to the user as a dot plot or, if there are more than 5000˜observations, a histogram, without the user needing to choose this first. A factor (refered to within iNZight as *categorical*) shows as a bar graph. This means the user focusses on exploring the data without the need to first *understand* the data.

In many other data analysis programs, graphs are created by the user first selecting the *type* of graph to display, and then choosing the variable. In an explorative sense, this makes little sense, as for example a variable called "age" might be numeric *or* categorical (for example age groups). The basic types of graphs available in 'iNZight' are shown in table 2. Table 3 shows "large sample" alternatives which are used when sample sizes exceed 5000. The three basic plot types are *bar charts* for exploring categorical variables, *dot plots* for exploring a single numeric variable (and possibly its relationship with another categorical variable), and *scatter plots* for exploring the relationships between two numeric variables. *Histograms* are are more useful way of exploring the distribution of values a numeric variable takes when the sample size is large (that is, seeing individual points is no longer useful, nor aesthetically pleasing), while *hexagonally binned plots* are used to replace scatter plots in the same

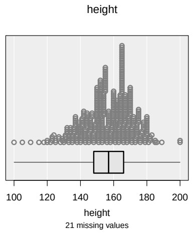
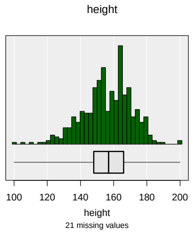
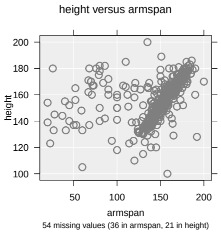
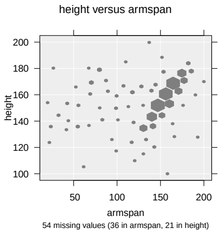
Table 2: iNZight default plot types are determined by the variable type of the first variable and, if specified, the type of the second variable.

Variable 1	Variable 2		
	None	Categorical	Numeric
Categorical			
Numeric			

situation, allowing users to see where the bulk of points are located. Of course, this is not the only way to look at large data, which I will get to shortly.

Another way of exploring relationships is by *subsetting* or *faceting*, which is easily accessed and indeed encouraged from the iNZight interface. The two subsetting variable slots allow users to subset the plot, which presents a slider to look at individual levels of the chosen variable, as shown in **??**. This is particularly advantageous to exploring trends *over time*, as users can use the "Play" button and watch as iNZight automatically plays over the range of levels (which might be "years" in a longitudinal dataset). This use of motion to explore trends in the data over time was most famously demonstated by Rosling (2010) in his BBC documentary "The Joy of Stats". However, time is not the only useful subsetting variable: many other categorical variables are often important when exploring relationships in a dataset, for example age and ethnicity.

Table 3: iNZight's alternative plots for large samples. Other plot types do not have a large sample alternative (i.e., bar charts).

Plot	Small	Large
dot plot		
scatter plot		

Of course, there is only so much one can learn from subsetting, and often more advanced visualisation techniques are required. For this, iNZight provides the *Add to Plot* system, which displays a list of options for the user to adjust, and most importantly only those related to the current plot. The most notable feature is *colour*, which can either be set to a single colour or, more usefully, coded to another variable in the dataset. This panel is also reactive, so a user's choices will open up some new options (or remove others), allowing them to explore the possibilities offered without having problems (for example trying to colour a bar chart by a numeric variable). Other codable features include *point size* and *point shape* (useful for printed media or for colourblind audiences).

Other options under the *Add to Plot* system include adjusting axes (by adding transformations or changing labels), adding additional information to specific plots (trend lines and smoothers to scatter plots, for example), and *locating points*. The last of these is an essential part of data exploration: often we want to explore outliers or else identify a specific point in the data. iNZight allows users to use a *point-and-click* identification method to highlight and label specific points, which are retained in future plots (so an individual can be tracked over different variables to explore them in detail), or points can be identified contextually, either by their specific value, or as an “extreme” value.

Since switching variables often results in a change of plot, iNZight remembers most settings (for example *colour*) so those variables remain visible in the dataset. This makes it easy to explore, for example, the effect of ethnicity on other variable relationships. Since some plots do not handle all arguments, iNZight only uses those that do: for example, switching to a two-way bar plot, the “colour” variable would be ignored without being forgotten; changing back to, say, a scatter plot reactivates the colour variable.

Visually assessing relationships in the dataset is generally the first step in a more in-depth analysis, and often we find that what we thought was an effect is actually not significant. iNZight provides some simple *visual inferencial markup* to plots that provide a visual way of exploring if a particular relationship might actually be there, or is simply a function of randomness. For example, a bar chart often hides the underlying population size, so the significance of differences in bar height is not obvious. By adding error bars, however, we can begin to make inferential calls about those differences. Another example is the trend line on a scatter plot: the sample estimate might display an upwards trend, but this may be affected by just a few points. By adding a sample of *bootstrap trend curves*, as shown in fig. 1, we can assess how *real* this relationship might be.

Another type of *visual inference* iNZight offers is *comparison intervals*, estimated using the ‘iNZightMR’ package (Elliott et al., 2020d). These intervals provide adjusted intervals designed primarily for visual comparison. In the two-way bar chart shown in fig. 2, the *within group* comparison intervals can be

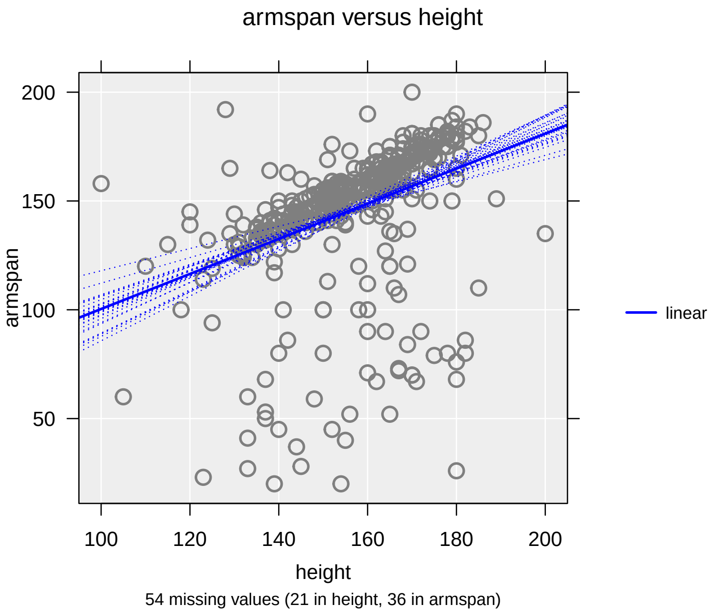


Figure 1: A scatter plot with a linear trend, showing a bootstrap sample of trend curves.

interpreted as “if the intervals do not overlap, then for a particular level of ‘travel’ there may be a different between the proportions for each ‘gender’ ”; in the figure, the appears to be a difference between the proportion of males and females who travel by bike, but otherwise the distribution of travel is the same. It is important to note that these *are not* confidence intervals: they account for multiple comparisons and the covariance between variables.

Having done as must visual exploration as iNZight provides, it is likely that some numerical information might be desired. For example, when discussing the data it is useful to have summary statistics, such as the mean or quantiles. iNZight provides a “Get Summary” button, which provides a simple summary of the currently displayed graph. Like the plots, the types of summaries depends on the chosen variable(s) and, in some cases, additional features (scatter plots provide extra information about the basic relationship of any fitted trend lines,

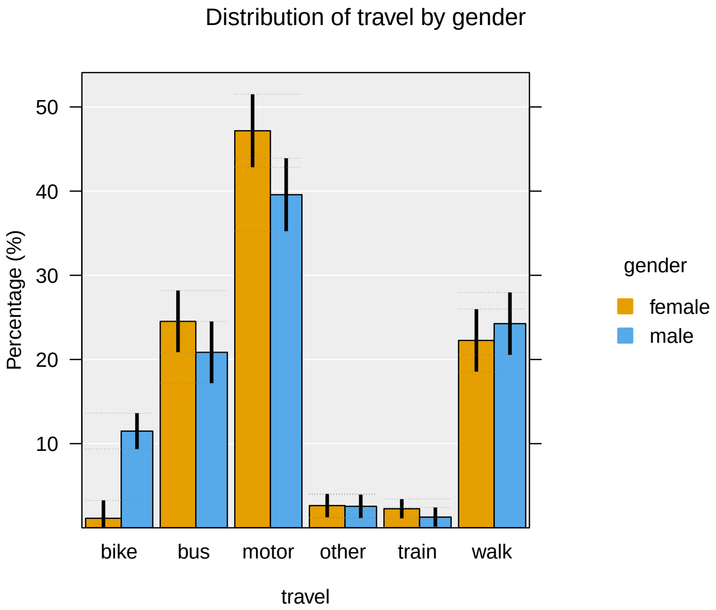
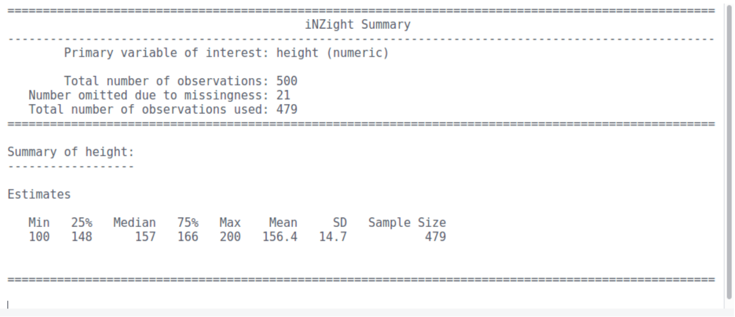


Figure 2: A two-way bar chart with comparison intervals.

for example). The primary use is to extract values from the graph, with a few extra variables (such as variance) which are considered simple summaries. The philosophy remains "look first". A simple summary of a dotplot is given in fig. 3.

Once visual and numeric exploration has been complete, the next step is often to pose and test hypotheses. Again, iNZight provides this functionality through the simple "Get Inference" button, and works similarly to "Get Summary", but with a few slight differences. The type of inference information and hypothesis tests available depend on the chosen variables, though users are also presented with some controls to specify information about their hypothesis. Additionally, *bootstrap inference* is provided which uses bootstrap sampling using the 'boot' package (Canty and Ripley, 2020) to obtain inferences. The basic inferences include confidence intervals for most parameters, as well as *p*-values of differences (in a two-way table of counts, for example).



```
====================================================================================================
                                          iNZight Summary
----------------------------------------------------------------------------------------------------
        Primary variable of interest: height (numeric)

        Total number of observations: 500
   Number omitted due to missingness: 21
   Total number of observations used: 479
====================================================================================================

Summary of height:
------------------

Estimates

   Min   25%   Median   75%   Max    Mean     SD   Sample Size
   100   148      157   166   200   156.4   14.7           479


====================================================================================================
```

Figure 3: The "Get Summary" window for a numeric variable ('height').

Table 4: iNZight hypothesis test options.

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

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

The choices of hypothesis test depend also on the variable types. For a sole numeric variable, a one-sample t-test is available in which the hypothesis value can be specified, and either a one- or two-sided test carried out. Table 4 shows the types of hypothesis tests available from iNZight.

This concludes the main functionality of iNZight for beginners: the ablity to load a data set and instantly view graphs and summary information, and perform hypothesis tests without having to understand how these fit together. iNZight provides a platform with with learners can explore on their own, and learn as they go.

4.3 Saving code history

As a statistics and data science learning tool, iNZight not only provides a way to learn data explorationg and visualisation, but also has begun a process of teaching basic coding through provision of R code scripts and snippets. All of iNZight's functionality is driven by calls to R functions, for which we can save the code for users to view later. This provides two things: first, a 'transcript' of a user's session, which is an essential part of *reproducible research* and allows sharing of their results; second it provides them with a script they can copy, paste, and edit to run in their own R session.

The code history script is a simple method of providing code, but it lacks *interactivity*. However, the latest version of iNZight (4.0.0) introduces a (developmental) "code widget" which gives the R code for the current plot. This has a bi-directional link with the GUI: when the user changes settings in the GUI, the code updates to reflect the change; but also if the user chooses to modify the *code* and run it, then the GUI controls are updated. This allows users to get a feel for writing code and setting how it affects the result.

4.3.1 Saving an R script

All of iNZight's methods call functions in other packages, many of which are themselves 'wrapper' functions. As an example, the Filter Data window in iNZight calls to the `filterData()` function in the 'iNZightTools' package (Elliott et al., 2020a), which returns the 'tidyverse' (Wickham et al., 2019) code used to perform the filtering steps. The code is extracted as follows:

```r
data.filtered <- iNZightTools::filterData(iris, "Species", "Setosa"
    )
cat(code(data.filtered))
# iris %>% dplyr::filter(Species == "Setosa")
```

The iNZight GUI simply fetches the code attached to each command it calls and appends it to the R history script. One main advantage of this, particularly using wrapper functions, is that the GUI does not need to know anything about the code—only that it exists. Therefore, improvements or changes could be made to the wrapper function, and these would be reflected in the resulting script without any changes to the GUI.

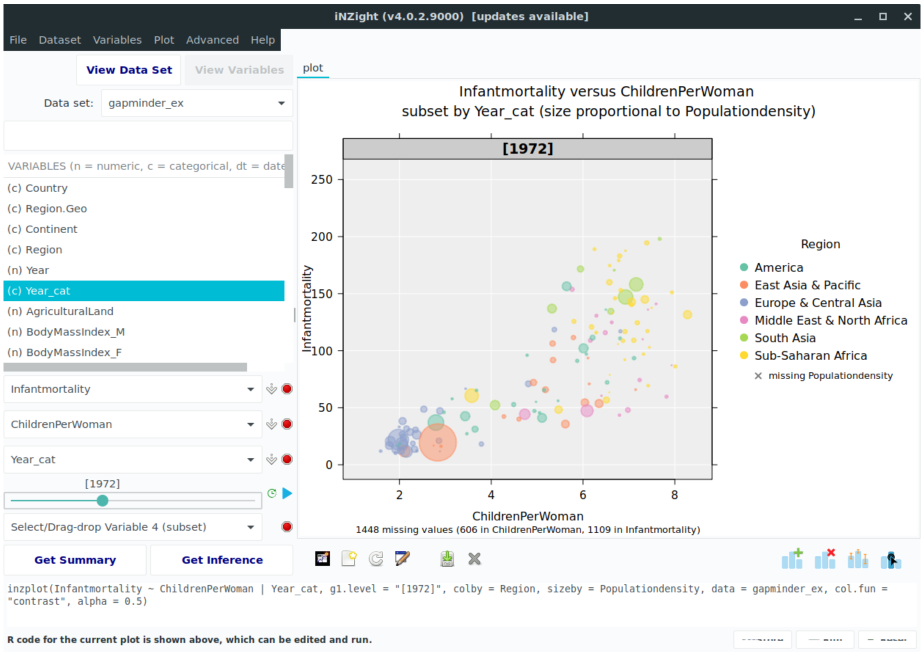
4.3.2 Live code editing

A more complex relationship exists between the 'code panel' widget, visible by turning on developmental features from the preferences menu. This module is displayed at the bottom of the window, as shown in fig. 4. Changing the configuration from the variables or the Add to Plot panel will update the code automatically, and vice verca.

To make this work, iNZight stores all of its settings in a single `list()`, in which each item corresponds to an argument passed to the `inzplot()` function (in the 'iNZightPlots' package (Elliott et al., 2020c)). From the `defaults()` function, we can obtain the list of 'default' values, and remove those settings that the user has not changed. These are then parsed together to construct the code call.

The reverse involves a little more work, since the code call needs to be converted into a `list()` object first. Then, it is compared to the current list and any changes are converted into actions in the interface. For example, if the user changes the value of `g1.level` in fig. 4, the value of the slider will be adjusted accordingly after comparing the change to the settings list.

The limitation of this approach is that it only works for the `inzplot()` function, and only accepts arguments the GUI has been coded to handle (any extra arguments are simply ignored). However, it does provide a very simple, elegant interface for users already familiar with 'iNZight' to begin exploring how



Data set: gapminder_ex

Infantmortality

ChildrenPerWoman

Year_cat

[1972]

```
inzplot(Infantmortality ~ ChildrenPerWoman | Year_cat, g1.level = "[1972]", colby = Region, sizeby = Populationdensity, data = gapminder_ex, col.fun =
"contrast", alpha = 0.5)
```

R code for the current plot is shown above, which can be edited and run.

Figure 4: The code panel shown at the bottom of the iNZight window describes all of the user's chosen settings for the current graph.

code works.

4.4 Analysing surveys with iNZight

Survey data is common in official statistics, and has been fully integrated into 'iNZight' to allow researchers to explore data from surveys with as much ease as they would standard data. To do so, 'iNZight' requires the survey information to be specified once, at which point all future actions will be performed using the survey design. This includes graphics, summaries and inference, data manipulation, and code writing. The most common survey structures are supported: stratified and cluster based surveys, replicate weights, as well as post-stratification or calibration of surveys using additional data.

There are three ways of specifying survey information to 'iNZight'. The first uses a simple dialog window asking the user to input the information manually, as shown in fig. 5. This allows users to specify the design of a survey if they already know it, and provides a straightforward interface to do so. However, it does make it difficult for learners and researchers not as familiar with the design to get started with their exploration. To get around this, we added two new methods for specifying survey information from a separate file.

The first of the two methods again uses the survey specification window, but allows users to locate a pre-configured file containing the survey design information in a specific format. 'iNZight' reads this file and applies the information to the existing dataset. The last method makes it even simpler, and only requires users to import one single file: the survey design specification (so long as the data is in the same directory). This is imported through 'iNZight's standard "Import Data" interface, and automatically loads the data and sets up the design all in one single step.

some stuff about the format of the survey spec file.

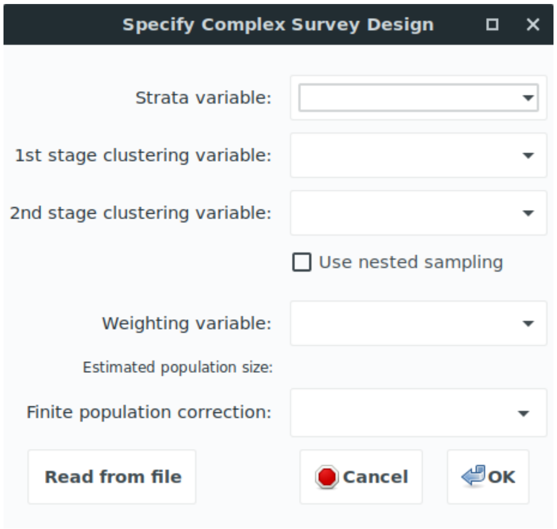

Specify Complex Survey Design
Strata variable:
1st stage clustering variable:
2nd stage clustering variable:
□ Use nested sampling
Weighting variable:
Estimated population size:
Finite population correction:
Read from file
Cancel
OK

Figure 5: Specifying survey information in the survey specification window.

Having specified survey design information, users will, in most situations, not have to worry about any other changes to their iNZight workflow. Graphs will automatically take into account the survey information, summaries and inferences will too. Even data manipulation switches over to survey-counterparts, and the code written to the R script remains valid.

Behind the scenes, iNZight passes around a survey object, which external helper functions (in the 'iNZightTools' package, for example) can expect. Many of the manipulation methods use functionality from the 'srvyr' package (Freedman Ellis and Schneider, 2020) to provide 'dplyr'-like syntax (Wickham et al., 2020), and minimises special handling of survey data. In cases where the syntax is not possible, we simply use `update()` to add, manipulate, or remove variables within the survey design.

When iNZight recieves a new survey design object, it extracts the data and displays this to the user. In this way, iNZight can natively pass survey objects around any displays the information accordingly.

4.5 Other modules

The main iNZight window focusses on simple exploration and analysis of a standard dataset. However, there are many situations where a dataset may have unique features (such as time series or map data) or a user might want to perform some more complex operations (model fitting, for example). Rather than attempting to integrate these into the main iNZight program—which would quickly become unwieldy—we created individual *modules*, accessible from the **Advanced** menu. Here, we overview the structure of these modules, and demonstrate a few of them.

4.5.1 The structure of iNZight's modules

Similarly to the components discussed in section 3, each of the modules is itself a Reference Class object, contained within the 'iNZightModules' packages (Elliott and Barnett, 2020). These modules are able to call a helper function of the main GUI object, allowing them to replace the left-hand-side control panel.

Each module is designed with a singular data type or statistical method in mind. For example, the *Time Series* module requires time-based data, and provides functionality to draw, decompose, and forecast time series data. Being a *child* of the main GUI, these modules have access to the dataset imported by the user, as well as the ability to change the menu items and plot toolbar to add custom buttons.

Module layout is typically constructed in the same way an analysis script might be, from top to bottom. The user starts out be choosing variables of interest, or providing necessary information (such as a time variable) to the module. Then, new options are activated, logically stepping the user through the process of exploring their data. In some, such a the Model Fitting module, various sections of the interface are containing within expandable frames to keep the interface uncluttered. However, firmly at the center of all the modules, is *graphics*: a plot is displayed as soon as possible, and updated as the user changes various inputs.

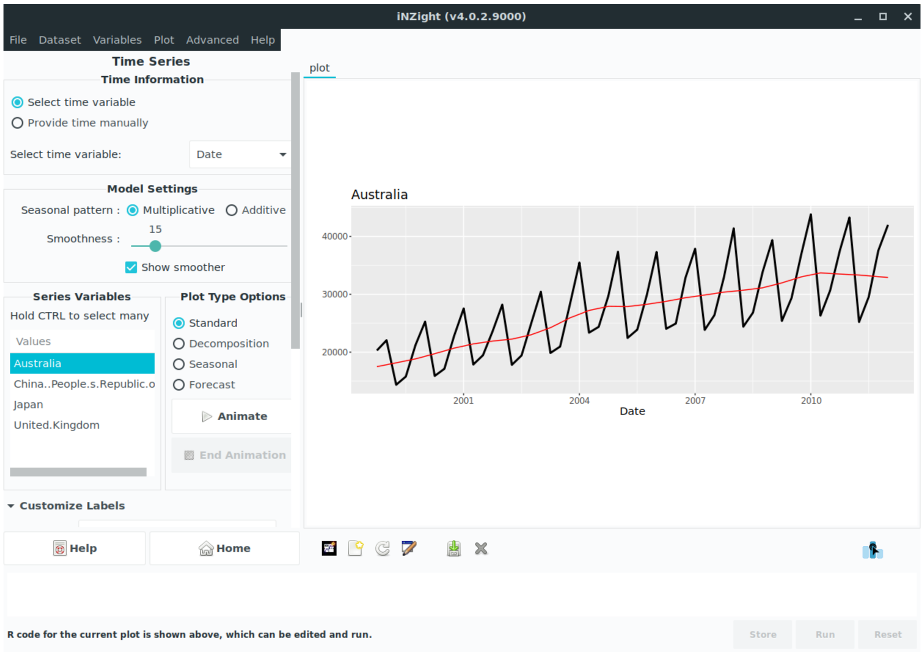
Within each module, there are usually one or two calls to external 'wrapper functions', either other packages maintained by the 'iNZight' team (for example, the Time Series module calls functions in the 'iNZightTS' package), or elsewhere. These packages are, however, also designed to be standalone packages, once again in an attempt to bridge the gap between high- and low-level code. These packages provide simple wrappers for some complex actions, which are user-visible and may, in time, be accessible through the interface (similarly to the plot code in the main iNZight interface).

4.5.2 Built-in modules

iNZight is shipped with several advanced modules, detailed below. These provide visualisation and analysis functionality for some common data types and processes(?). Here we give a quick overview of the functionality provided by each.

Time Series A common type of data available, and of interested, to users is time series data, so of course iNZight provides a module dedicated to visualising these types of data. When loaded, the module attempts to detect a 'time' variable, and otherwise asks the user to provide one. Alternatively, users may manually provide time information, for example if the information is not available or formatted incorrectly. Once done, iNZight displays a graph of the first variable in the dataset over time, as shown using the "Quarterly New Zealand visitors" data in fig. 6.

Users can choose between multiplicative (the default) and additive models, and have prominent control over the 'smoothness' of the trend line. This al-



● Select time variable

○ Provide time manually

Select time variable: Date

Seasonal pattern : ● Multiplicative ○ Additive

Smoothness : 15

☑ Show smoother

Australia

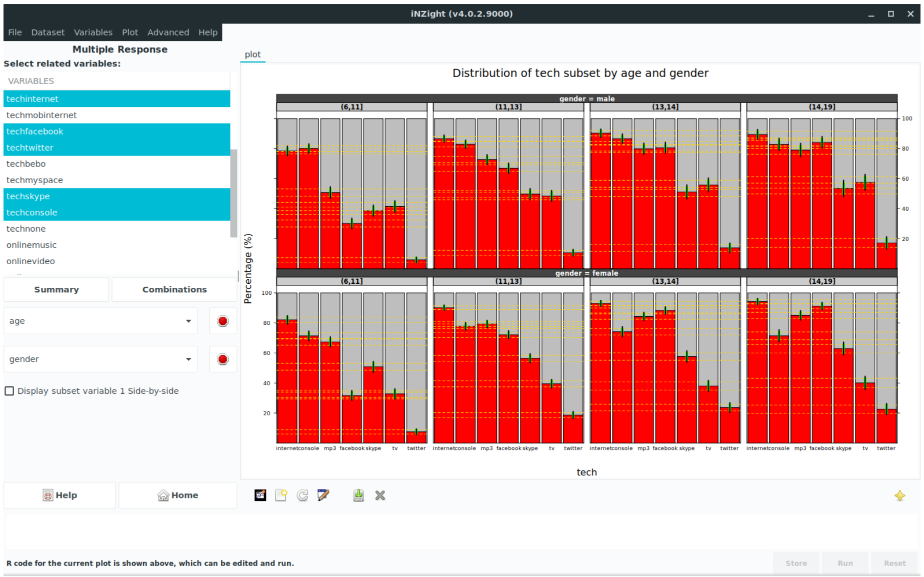
● Standard

R code for the current plot is shown above, which can be edited and run.

Figure 6: Visitors data.

lows users to see how results can change based on the level of smoothing. The available variables are presented, and can either be displayed one-at-a-time, or multiple-selected to compare series. The seasonal-trend decomposition using LOESS (STL) algorithm (Cleveland et al., 1990) is used to provide the decomposition and seasonal graphs, and Holt-Winters' algorithm (Holt, 2004; Winters, 1960) is used for forecasting.

Multiple Response A seldom discussed data type is *multiple response*, which typically come from surveys with 'tick all that apply' type answers. The 'iNZightMR' package (Elliott et al., 2020d) provides methods for analysing these types of data, such as calculating the proportion of responses in each category with the correct standard errors. The *Multiple Response* module connects to the aforementioned package, and when loaded presents the user with a list of 'binary'



Multiple Response

Select related variables:

VARIABLES
techinternet
techmobinternet
techfacebook
techtwitter
techbebo
techmyspace
techskype
techconsole
technone
onlinemusic
onlinevideo

age

gender

☐ Display subset variable 1 Side-by-side

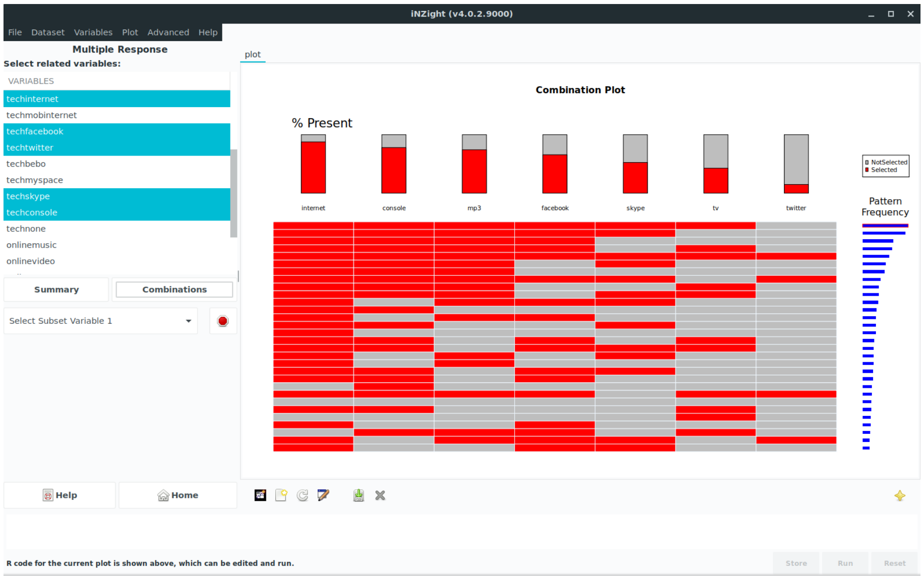
R code for the current plot is shown above, which can be edited and run.

Figure 7: Student responses to having using various technologies.

variables in the dataset (that is, with a 0/1 or 'yes'/'no' value). These can be multiple-selected and a graph is displayed.

In the example of fig. 7, the responses of students to a questionnaire about what technologies they have used is shown, subset by age and gender. The error bars account for multiple responses, and the horizontal yellow lines are 'comparison intervals', which indicate a difference between groups if they do not overlap. Another graph available is the "Combinations" plot, showing the frequency of various combinations of responses (fig. 8). This allows users to explore the frequency of response combinations.

Maps A hugely popular type of data is geographical data in a huge variety of forms. iNZight's Map module allows users to plot location data (i..e, longitude/latitude coordinates on a map) or regional data (countries, regions, etc). For point-type data, these are effectively a scatter plot with a map underneath, and include much of the same functionality: subsetting, colour, resizing, and



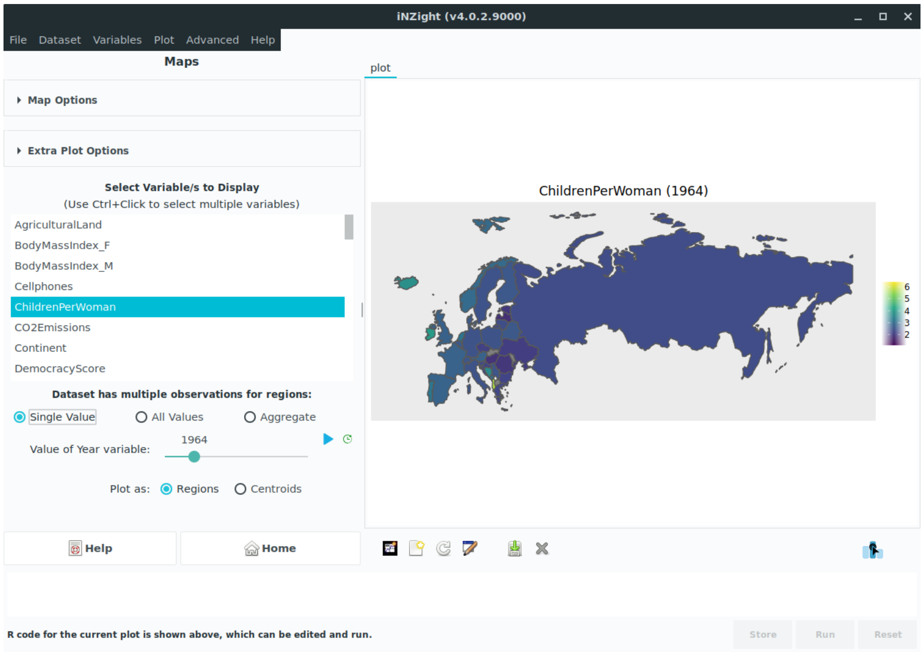
R code for the current plot is shown above, which can be edited and run.

Figure 8: Combinations of responses as to having used various technologies.

so on. Regions are different, however, in that they require *shape files*, which describe the boundaries of various regions. For example, the world shape file includes shapes of countries.

The difficulty with shape data is merging the shapes to the original data, since some countries are commonly spelled differenctly or using a range of different country codes. The iNZight Maps module automatically tries to guess the best match, and allows users to override this to improve matches, or manually match unmatched countries. It also lets users choose between a variety of shape files, including world maps, region maps (europe, america, etc), and some other specialised maps (for example New Zealand electoral districts). User may also import their own shapefile, if desired.

Once imported, variables can be chosen to graph on the map, in which case regions are coloured by the variable. Categorical variables are given one colour per category, while numeric are given a continuous colour palette and coloured accordingly. Additionally, a *time* variable can be included, in which case sum-



Select Variable/s to Display
(Use Ctrl+Click to select multiple variables)

AgriculturalLand
BodyMassIndex_F
BodyMassIndex_M
Cellphones
ChildrenPerWoman
CO2Emissions
Continent
DemocracyScore

Dataset has multiple observations for regions:

● Single Value ○ All Values ○ Aggregate

Value of Year variable: 1964

Plot as: ● Regions ○ Centroids

R code for the current plot is shown above, which can be edited and run.

Figure 9: Sparklines on a map.

maries can be graphed (e.g., mean, median, maximum, etc) or a mini time series "spark line" graph can be drawn on each region. This is demonstrated in fig. 9.

Model Fitting The final module built in to iNZight provides arguably the most used methods of statistics: regression modelling. This module provides normal, binomial, and Poisson modelling of a single response variable, and users may add variables as predictors either by drag-and-drop or double-cliking. The residual plot and summary output are updated instantly, allowing users to build up a model interactively. Additional summary plots are available, such as Cooks' distance, QQ-plots, and histograms. iNZight Model Fitting uses graphs provided by the 'iNZightRegression' package (Elliott et al., 2020b). A novel contribution is the display of bootstrap smoothers on graphs, as well as *bootstrap QQ-plots* and a *boostrap array of histograms*, allowing users to visually compare what the residual distribution might look like (i.e., under small sample sizes, non-normality might be due to chance). Another feature of the Model Fitting module is saving and later comparing (either via AIC or BIC) alternative models.

4.5.3 Addon modules

Updating modules within a package is a drawn-out process, and requires the package maintainer to sort out updates and release. To this, we recently began trialing a new "Add-on" system, allowing individual modules (each a single R file containing a reference class object) to be added, updated, and removed as needed. In this way, individuals—be it members of the 'iNZight' team, teachers of various statistics courses, or researchers—can develop their own modules and install these into iNZight. By using Github pull requests, they may have their work integrated into the 'official' iNZight addon repository, or they can distribute a file to their users who can install the module from a local file. This kind of structure is seen in many other programs, for example Excel and Notepad++ (**?**).

One exciting new addon being developed is the 'Bayesian demographic modelling' module, which connects to a set of packages used to perform the type of analysis used by Zhang et al. (2019). This work will provide small communities, demographics, and nations to explore and analyse their own data using a tool with which they are either already familiar, or has a very easy learning curve: iNZight.

5 Distributing and running iNZight

Deploying software is a difficult process, particularly when your users use a range of operating systems. Fortunately, since iNZight is essentially just a collection of R packages, it's very simple to install since R itself is already distributed for the most popular operating systems. However, many of our users are students with little software knowledge, so a simpler method is necessary. Note also that some of our R packages are not available on CRAN, so we need a way for those to be available also.

The first issue is to make the R packages we develop available online. The first choice would be CRAN, and at the time of writing we have [four or five] packages accepted. The main GUI packages 'iNZight' and 'iNZightModules', however, require a significant amount of work to ensure they adhere to the CRAN repository policies: they mostly center around reading and writing to the disk and saving of users' preferences. An alternative is to self-host the packages, which has two advantages:

1. we can upload packages instantly, and
2. updates and bug fixes can be released frequently, whereas CRAN suggests limiting updates to every 2–3 months.

We have a cloud-based repository using Amazon Web Services (AWS) at `https://r.docker.stat.auckland.ac.nz`. The "docker" component of the URL is there only for historic reasons, and will one day be updated.

Given packages available (details later) we have a distribution system available for Windows allowing users to install iNZight as a standalone program. This works because, on Windows, R can be installed to any directory of choice, and runs without requiring access to any other part of the filesystem. Therefore we bundle the R program, the R package library (including iNZight R package and its dependencies) and several other files to make it possible to launch iNZight via R with a simple double-click.

The build process requires compilation of all the package binaries for Windows, structuring of the iNZightVIT folder, building of the EXE installer, and finally deployment to the server for users to download. GitHub Actions (`https://github.com/features/actions`) is a useful automation framework which we now use to automate the entire build and deploy process for 'iNZight'. Using several individual *jobs*, the automation builds the latest package binaries and uploads them—along with the source files—to our R package repository. Then it builds the windows install directory with these binaries, and uses NSIS (NSIS Team and NSIS Community, 2020) to create an executable windows installer, which is uploaded to the remote server.

Obviously, the CRAN releases are managed manually and less frequently.

5.1 A note for macOS and Linux users

iNZight uses GTK as the window management system, however in recent years support for GTK on macOS has been neglected, and so we can no longer provide an official release of iNZight for Mac users. It is, however, possible to compile the necessary package dependencies manually and get it running, but this is not something we expect our users to do.

On Linux, support varies by distribution. Ubuntu, Debian, and Fedora-based systems natively support R, and GTK binaries are available, so it is easy enough to install the dependencies and R packages. However, to help, we provide a simple GNU Make script to install iNZight and create launch scripts, which may be installed onto the system to make `inzight` available from anywhere.

We do provide an online version of iNZight, called iNZight Lite, that provides much of the same functionality but runs in the user's web browser and connects to a remote R server.

- ✓GTK support for mac is dead - so we cannot support macOS any longer
- ✓Linux requires installation of some platform-specific packages (gtk, xorg, etc etc)

- ✓just like installing any other R package
- ✓but we do include a build tool to help install and set-up run/update scripts to make launching easier

5.2 Manual installation

Installing iNZight manually is usually a simple process. On Windows, no dependencies are required beforehand; on macOS and Linux, users must first install GTK and various other platform-dependent dependencies. And R of course. Then it’s a matter of starting R and installing from our repository:

```
install.packages(’iNZight’,
    dependencies = TRUE,
    repos = c(’https://r.docker.stat.auckland.ac.nz’,
        ’https://cran.rstudio.com’),
    Ncpus = 4L # recommended on Linux
)
```

Afterwards, iNZight can be run by loading the library and invoking the function of the same name:

```
library(iNZight)
iNZight()
# optionally pass a dataset directly:
iNZight(iris)
```

6 Discussion and Future Work

- iNZight is a simple-to-use tool for exploring/visualising/analysing data
- built for beginners with capabilities to progress to R-driven coding
- easy-to-learn makes it ideal for organisations that need to do specific jobs rarely (i.e., every few months) - more complicated tools are easy to forget

- flexibility of design makes it easy to add/change functionality as demand changes/grows (e.g., survey design)

6.1 Future Work

- better survey handling
- filling the ‘code writing’ gaps
- connecting to databases (and potentially processing as much as possible within the db)
- Te Reo translation (and flexibility to add more)
- . . .

Acronyms

API application programming interface. 4

AWS Amazon Web Services. 28

GUI graphical user interface. 1, 3–5, 8, 16, 17, 28

OOP object oriented programming. 4

OS operating system. 3

STL seasonal-trend decomposition using LOESS. 23

References

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

Cleveland, R. B., Cleveland, W. S., McRae, J. E., , and Terpenning, I. J. (1990). Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–33.

Elliott, T. and Barnett, D. (2020). *iNZightModules: iNZight GUI Modules*. R package version 2.5.4.

Elliott, T., Jin, O., He, Y., and Barnett, D. (2020a). *iNZightTools: Tools for 'iNZight'*. Available from `https://cran.r-project.org/package=iNZightTools`.

Elliott, T., Potter, S., and Banks, D. (2020b). *iNZightRegression: Tools for Exploring Regression Models with 'iNZight'*. R package version 1.3.0.

Elliott, T., Soh, Y. H., and Daniel Barnett (2020c). *iNZightPlots: Graphical Tools for Exploring Data with 'iNZight'*. Available from `https://CRAN.R-project.org/package=iNZightPlots`.

Elliott, T., Zeng, J., and Potter, S. (2020d). *iNZightMR: Tools for Exploring Multiple Response Data*. R package version 2.2.5.

Fox, J. (2005). The R Commander: A basic statistics graphical user interface to R. *Journal of Statistical Software*, 14(9):1–42.

Freedman Ellis, G. and Schneider, B. (2020). *srvyr: 'dplyr'-Like Syntax for Summary Statistics of Survey Data*. R package version 1.0.0.

Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10.

Lawrence, M. and Temple Lang, D. (2010). RGtk2: A graphical user interface toolkit for R. *Journal of Statistical Software*, 37(8):1–52.

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

NSIS Team and NSIS Community (2020). *Nullsoft Scriptable Installer System (NSIS)*. https://nsis.sourceforge.io.

R Core Team (2020a). *foreign: Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...* R package version 0.8-76.

R Core Team (2020b). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rosling, H. (2010). *The Joy of Stats*. [BBC Video]. Available from https://www.gapminder.org/videos/the-joy-of-stats/.

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

The jamovi project (2020). https://www.jamovi.org.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Verzani, J. (2019). *gWidgets2: Rewrite of gWidgets API for Simplified GUI Construction*. R package version 1.0-8.

Verzani, J. (2020). *gWidgets2RGtk2: Implementation of gWidgets2 for the RGtk2 Package*. R package version 1.0-7.1.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., and Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686.

Wickham, H. and Bryan, J. (2019). *readxl: Read Excel Files.* R package version 1.3.1.

Wickham, H., François, R., Henry, L., and Müller, K. (2020). *dplyr: A Grammar of Data Manipulation.* R package version 1.0.2.

Wickham, H. and Grolemund, G. (2017). *R for Data Science.* O'Reilly Media.

Wickham, H., Hester, J., and Francois, R. (2018). *readr: Read Rectangular Text Data.* R package version 1.3.1.

Wills, J. (2020). *NZGrapher.* `https://grapher.jake4maths.com`.

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342.

Zhang, J., Bryant, J., and Nissen, K. (2019). Bayesian small area demography. *Survey Methodology*, 45(1).

Acknowledgements

iNZight is a free to use, open source software. The work would not have been possible without the support of the University of Auckland, Census at School, ..., Statistics New Zealand, and the Australian Bureau of Statistics.