Journal of Statistical Software

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

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

Tom Elliott
Victoria University of Wellington

Chris Wild
University of Auckland

Daniel Barnett
University of Auckland

Andrew Sporle
University of Auckland

Abstract

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

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

1. Introduction

The R programming environment (R Core Team 2020) is used throughout statistics and data science due to it being open source, easy to learn, and backed by a huge package repository to solve even the most unique of problems.

- several graphical user interfaces (GUIs) have been developed over the years to ease

access to advanced features of R: graphs, hypothesis tests, etc

- eg: Fox (2005), Fellows (2012)
- these tend to work by asking users to first choose an action, then fill in the fields (including variable choice)
- this requires some higher level of understanding to get useful information out (i.e., need to know what a "t-test" is)

An alternative approach is to work variable-first, such that users choose variables they are interested in, and then choose from an automatically curated list of options to perform. **iNZight** uses this approach, and presents users with a exploration-focussed interface.

- focus is on visualising, removing the need for any basic understanding of statistical procedures to get started with data visualisation
- like other GUIs, there's a code component: however, **iNZight**'s is more *behind-the-scenes*; useful for seeing a history of what you've done, with a little emphasis on editing a command to see how it changes things
- goal is to develop data exploration skills before moving them on to coding (high-level to low-level)

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

2. A tour of iNZight's features

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

2.1. Loading data

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

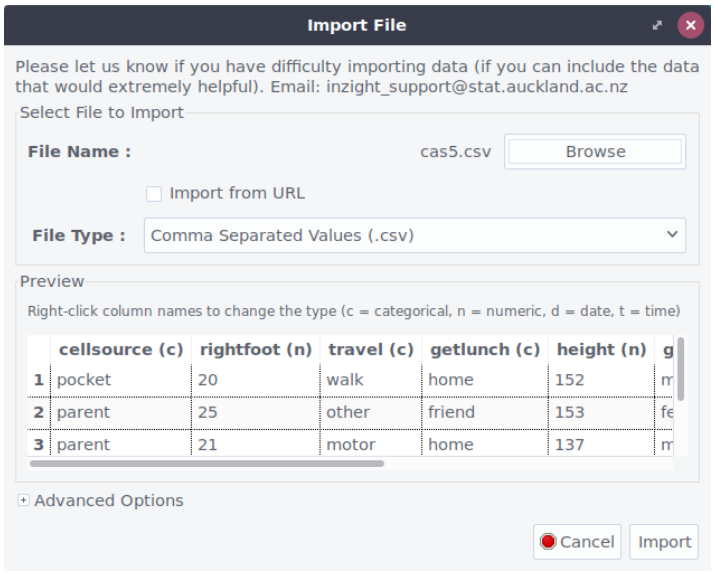

Figure 1: Load Data window, showing the chosen file, the File Type (guessed from the extension), and a preview of the data.

dedicated to reading specific file formats. However, users must still know the correct package and function for their files. **iNZight** provides a simple Load Data window from which they can select a file to import. The software detects the file type from the file extension and attempts to read the file. If successfull, a preview is displayed for the user to check before proceeding to import the file. Figure 1 shows the Load Data window, which has detected a comma separated values (CSV) file and used the appropriate function in the background. Currently, **iNZight** supports files in CSV, tab-delimited, Excel, SAS, Stata, SPSS, R-data, and JSON formats.

In addition to the preview, **iNZight** also has an *Advanced Options* section for some specific formats (currently only CSV files). Namely, this allows users to override the default delimiter, for example in European countries where the semi-colon (;) is used (the comma is reserved as the decimal separator), as well as different encoding formats. The preview is updated when these options are changed, meaning users do not have to know specifically what they need, and can quickly trial-and-error until the preview looks right.

2.2. Creating graphs

Within **iNZight**, graphics are at the core of the user experience. The very first prototype included a drag-and-drop of variables to create a plot, and that is how things have remained. Behind the scenes, **iNZight** detects the variable types (numeric, categorical, of a date-time) and draws an appropriate graph. For example, a single numeric variable produces a dot plot, while a categorical variable produces a bar chart. The user does not need to know what type of graph they want to create from a chosen variable, allowing them to freely explore the dataset by drag-and-drop of variables onto the Variable boxes (or using the drop down menus).

The first variable box is referred to as the *Primary Variable of Interest*. That is, this is the

Figure 2: Demonstration of plot modifcations available from **iNZight**'s ADD TO PLOT menu.

variable we want to know about, and how it is influenced by other variables. For example, if 'height' is chosen as Variable 1, and we want to know how 'height' changes with ethnicity or age. In the latter case, the result will be a scatter plot *with 'height' on the y-axis.* In addition to the first two variable slots, **iNZight** includes two Subset variables to quickly and easily facet the plot and explore more complex relationships.

Finally, there is an entire panel dedicated to plot modifications: ADD TO PLOT. This is accessed either from the PLOT menu, or from the button in the PLOT TOOLBAR. From here, users can choose from a selection of alternative plot types (limited by the types of variables), as well as choose a colour variable, sizing variable, plot symbols, trend lines, changing axis labels and limits, and much more. The options are presented in an interactive format such that the graph updates whenever the user changes values, allowing them to explore "what happens if . . . ", and "what does this do?". The goal is to allow beginners to explore a dataset without being limited by having to learn how to do a wide range of actions. Figure 2 shows a graph produced by **iNZight** exploring the relationship between infant mortality and GDP, region, population, and year.

2.3. Summaries and inference

To supplement the visual graphics, **iNZight** also provides two textual output modes: *summaries* and *inference*, accessed from the GET SUMMARY and GET INFERENCE buttons respec-

tively. Summary information includes basic information about the chosen variables, including mean, standard deviation, quantiles, and so on, acting as a quick reference for values that are likely estimable from the graph itself.

The inference information provides confidence intervals for quantities such as means and proportions, and additionally provides a simple interface for performing hypothesis tests. **iNZight** displays a selection of tests available for the chosen variable(s), as shown in TABLE. The inference information can either be calculated using Normal theory or bootstrap methods (using the **boot** package, Canty and Ripley 2020).

2.4. Data wrangling

Usually the first thing researchers want to do when they first begin exploring a dataset is to create a set of exploratory graphs. However, it is often not possible to get the correct graphs from the raw data alone. Applying transformations and other modifications to the data can allow researchers to explore the data correctly, or explore it from a different perspective. **iNZight** contains two *data manipulation* menus: Data and Variables. The former acts on the data set as a whole, while the later modifies individual columns.

In their book *R for Data Science*, Wickham and Grolemund (2017) describe many data manipulation methods including filtering, aggregation, and reshaping. They provide the **tidyverse** (Wickham *et al.* 2019) code for these actions, which **iNZight** uses behind-the-scenes to implement the behaviours. However, **iNZight** provides a GUI interface to these (often complex) methods, allowing users to quickly and easily filter by value, convert from *wide* to *long* form, or merge two related datasets together. In most cases, the interface allows users to fill out the fields which change according to previous selections, and at the bottom is a preview of what the data will look like, as demonstrated in FIGURE.

Supplementary to the dataset operations, the Variable menu provides a selection of variable transformation and modification actions. For example, numeric variables can be converted to categorical (a common example is `Year`), or categoric variable levels can be renamed, reordered, and combined. Users also have the option of creating custom variables using R code, as well as renaming and deleting entire variables. In most cases, **iNZight** creates a *new* variable, for example converting `Year` to categorical might yield the variable `Year.cat`, which makes the experience more exploration-friendly.

2.5. Special data types

- a lot of data is simple rows of individual observations
- most standard graph types offered by **iNZight** can cover the basic explorations
- some cases where either the data has unique features that *must* be incorporated (e.g., surveys) *or* the data has unique constructs and needs a specific exploration techniques (time series, multiple response)
- **iNZight** provides capabilities for some of each (plus extensibility for more)

Complex survey designs

- native handling of surveys: set and forget
- stratified/cluster surveys; replicate weight surveys; post-stratification/calibration
- design stored and used in place of dataset: graphs, summaries, transformations all use survey-specific functions from the **survey** package (Lumley 2004).

Time series

- common type of data beginners may want to explore
- needs a time variable, either encoded correctly or can be coerced (or specified manually if known)
- time on the x axis, responses typically connected by lines to see change over time; can be one or multiple series
- seasonal decomposition: trend + seasonal effect + residuals (STL algorithm)
- teaching-focus: animations to show decomposition
- Holt-Winters forecasts Holt (2004); Winters (1960)
- used in NZ high schools for time series module

Maps

- another commonly seen data type
- either points on a map (e.g., a specialised scatter plot) or areas on a map, colour coded by variables
- **iNZight** can do both; regions requires data with location labels (countries, counties, regions, etc); points just a longitude and latitude variable
- lots of map-specific features: colour by region; aggregate time variable; plot mini plots over regions (e.g., barchart of categorical variable by region)
- the hard part is often figuring out shapefiles: **iNZight** lets user pick the type of map, and **iNZight** does the rest (with previews to help fine-tune)

Other types

- multiple response data
- multivariate data (addon module, Section 4)
- frequencies

- extensible to others too

2.6. Code writing for getting started with R

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

Another feature is the inclusion of the R code box at the bottom of the interface, highlighted in FIGURE. This displays the code used to generate the current plot and, more importantly, can be edited by the user and run. When the user does so, the interface detects changes in the code and applies those changes to the GUI, providing a seamless way for users to begin experimenting with code whilst retaining the familiarity of the GUI.

Users can also store the code for the current plot, which places it into the R script. This script contains a history of all actions from importing the data through transformations and manipulations. The goal is to provide a record of what the user did, as well as something they can copy and paste into R and run themselves, editing where desired. This allows R learners to explore a dataset with a GUI tool and make the unique actions necessary, and then generate a script which they can build on and learn from. A similar code box is displayed in the Get Summary and Get Inference windows, with plans to implement this behaviour throughtout **iNZight** in future.

The code used by **iNZight** uses **tidyverse** (Wickham *et al.* 2019) syntax, as this provides a simple introduction to R with many simpler and verb-like syntax for data wrangling, and is used the *R for Data Science* (Wickham and Grolemund 2017). To demonstrate the script produced by **iNZight**, APPENDIX contains the script generated during the tour presented in this section.

3. Technical details

The interface for **iNZight** is developed entirely within R, made possible by three main packages. **gWidgets2** (Verzani 2019) and **gWidgets2RGtk2** (Verzani 2020) provide a simple widget-based application programming interface (API) to building an cross-platform interface with R. The second package, **gWidgets2RGtk2**, is an interfacing package which provides a lower-level wrapper for the more complex functionality within the **RGtk2** package (Lawrence and Temple Lang 2010), which itself calls back to C libraries for the GTK+ (The GTK+ Team 2020) windowing system. Together, these packages provide a platform- and library-independent API for creating windows with uesr inputs from R.

The framework used to build **iNZight** is an object oriented programming (OOP) framework, which is well suited to GUIs, of which there are several within R. **iNZight** uses *reference classes*, which are also used by **gWidgets2**, to describe individual components of the interface. Each panel is a *class*, with individual buttons, methods (actions), and even smaller components

which are themselves classes. OOP also allows for *inheritance*, which allows developers to describe a general class which can be shared to several related components, but which may have different layouts or methods. FIGURE shows the **iNZight** GUI with some of the major class components annotated.

The structure of each class is, in most cases, a set of attributes that the user can control, stored as *properties* of the class. There is also a set of *methods* which can be used by the class to react to user input, or perform actions. Most components have a main *action* method, which performs the primary action of the component. For example, the '`iNZFilterData`' class contains a `filter_data()` method. These main functions collect the user's input and pass them to one or more *wrapper* functions; in the case of '`iNZFilterData`', one of the functions is `iNZightTools::filterNumeric()`. A skeleton example of the Filter Data window class is shown below:

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

In this oversimplified example, the user will be displayed a drop down `gcombobox()` to choose

a variable to filter on. When they click the button, the data will be filtered and passed back to the main GUI. The filtering uses the `switch()` function to select the wrapper based on the users selection (not shown to save space).

Each major component has a similiar structure, and calls to various functions, many of which come from wrappers in other **iNZight*** packages. For example, plots are generated by calls to `iNZightPlots::inzplot()`, while data import is handled by `iNZightTools::smart_read()`. This function uses the file extension to guess the file type and load the data using the appropriate methods. The wrappers enfore the desired structure, where the interface asks for the values of arguments to be passed, and keeps the data-specific logic out of the GUI.

A second advantage is that the individual wrapper functions can be designed to include the lower-level R code used to generate the result, which the GUI can fetch from the returned data and attach to the script. Here is an example of the result returned by `iNZightTools::smart_read()` (which can read from a remote URL):

```
R> library("iNZightTools")
R> data <- smart_read("nls.dta")
R> cat(code(data), sep = "\n")

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

The `iNZightTools::code()` function returns the R code attached to the resulting object, allowing a user to see that the **haven** package (Wickham and Miller 2020) was used to read this Stata file (`.dta`).

While the GUI packages provide the structure of the visual GUI, it's the collection of R packages developed alongside **iNZight** that are the powerhouses of the program. The main reason for creating separate packages was to force the separation of interface and data logic, but also to allow the parallel development of a separate interface (Section 5.4) using the same wrapper functions. The collection of packages within the **iNZight** project are described in TABLE.

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

3.1. Usage

At its core, **iNZight** is simply an R package that can be installed and run like any other, which is covered in Section 5. Once installed, the main program can be started by creating a new instance of the main GUI class '`iNZGUI`', as demonstrated below.

```
R> library("iNZight")
R> ui <- iNZGUI$new()
R> ui$initializeGui()
```

For most users, however, the simpler wrapper function `iNZight()` can be called instead. This can optionally take a `data` argument, which will launch **iNZight** with the data loaded and ready to explore.

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

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

4. The add-on system

Most users will likely find all they need with the main **iNZight** interface, able to explore and visualise their data quickly and easily. However, there are cases where, as mentioned earlier, specific types of analyses and graphics are required for some data types (time series and maps, as examples). Of course, the total range of data types available is close to limitless, and there are increasingly more R packages on CRAN every day providing new opportunities. Rather than requiring each new datatype or method to be manually coded into **iNZight** by the developers, we have crafted an *Add on* system allowing anyone to create their own **iNZight** modules.

Installing existing add-ons is easy. Users can either use the Module manager to add, update, and remove modules from our add-on repository, or from a custom URL or file. In all cases, the file is placed in the `modules` directory, which users (and developers) can also place files manually. All files in this directory are then displayed in the Advanced menu of **iNZight**, and when opened have access to the current data and other aspects of the interface.

The module files themselves describe a single class object which inherits from '`CustomModule`'. This parent class provides several methods, including the initialization of the module panel in the left-hand-side of the **iNZight** interface. Additional properties and methods can be written by the developers of individual modules. This opens up possibilities for teachers, research groups, or even R package developers themselves to write custom modules and distribute them to their desired audience.

As an example, FIGURE shows a prototype of an upcoming Bayesian demographic modelling (Zhang *et al.* 2019) module which will be used by small non-government organisations (NGOs) to do small-area estimation. For example, estimation of life expectancy in small ethnic groups. This opens up advanced methods that were previously only available to proficient coders, who may now perform complex modelling procedures themselves.

5. Installation and availability

- can install as an R package from combination of our repository and CRAN (still working towards publishing final few packages to CRAN)

5.1. Operating system specific requirements

- windows: will be prompted to install GTK binaries on first run
- macOS: very difficult, need to install XQuartz, gtk2+ framework, and compile **RGtk2** manually ...
- Linux: install system dependencies, then install as usual

5.2. Windows installer

- difficult to expect novice users/students to install R, etc
- we have a bundled version which comes as a `.exe` installer, unpackaging into chosen directory (default `Documents`
`iNZightVIT`) which includes a copy of R and package library
- also some shortcuts to launch R in a subdirectory containing a `.Rprofile` which loads **iNZight** and launches the GUI

Updating

- updater included with installer: makes it easy for non-R users to update packages periodically

5.3. Docker image

- Docker (**?**) lets developers bundle apps for distribution
- containiner based on Linux and works for Linux and macOS hosts
- less speedy (performance) but doesn't require huge install; currently the only way to run **iNZight** on macOS without manually compiling lots of things
- still requires a little work from the user to enable X11 forwarding (so the interface windows display on the host screen)

5.4. Online shiny version iNZight Lite

- for users who can't install (e.g., macOS, tablet)
- most of the same functionality, separate but parallel development
- call to same packages: same results across systems

- runs on Amazon Web Services (AWS) server: some performance issues

6. Summary and future work

- **iNZight** provides an easy-to-use GUI to simple exploration and visualisation tasks for statistics and data science students/beginners
- variable-first: doesn't require knowledge of techniques beforehand
- data manipulation techniques and other advanced data-specific features (e.g., surveys)
- code writing for those interested in migrating towards learning to code in R
- add-on system for easy expansion by course coordinators/researchers/R package maintainers who want to provide a simple interface to their package
- available on Windows and Linux

6.1. Future Work

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

Acknowledgments

iNZight is a free to use, open source software. The work would not have been possible without the support of the University of Auckland, Census at School, ..., Statistics New Zealand, and the Australian Bureau of Statistics. We also thank the technical support of the University of Auckland IT services for providing hosting services for our repository and Lite servers.

References

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

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.

Holt CC (2004). "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages." *International Journal of Forecasting*, **20**(1), 5–10. doi:https://doi.org/10.1016/j.ijforecast.2003.09.015.

Lawrence M, Temple Lang D (2010). "**RGtk2**: A Graphical User Interface Toolkit for R." *Journal of Statistical Software*, **37**(8), 1–52. URL http://www.jstatsoft.org/v37/i08/.

Lumley T (2004). "Analysis of Complex Survey Samples." *Journal of Statistical Software*, **9**(1), 1–19. R package verson 2.2.

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

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

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

Verzani J (2019). ***gWidgets2**: Rewrite of **gWidgets** API for Simplified GUI Construction*. R package version 1.0-8, URL https://CRAN.R-project.org/package=gWidgets2.

Verzani J (2020). ***gWidgets2RGtk2**: Implementation of **gWidgets2** for the **RGtk2** Package*. R package version 1.0-7.1, URL https://github.com/jverzani/gWidgets2RGtk2.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the **tidyverse**." *Journal of Open Source Software*, **4**(43), 1686. doi:10.21105/joss.01686.

Wickham H, Grolemund G (2017). *R for Data Science*. O'Reilly Media. ISBN 978-1491910399. URL https://r4ds.had.co.nz/.

Wickham H, Miller E (2020). ***haven**: Import and Export SPSS, Stata and SAS Files*. R package version 2.3.1, URL https://CRAN.R-project.org/package=haven.

Winters PR (1960). "Forecasting Sales by Exponentially Weighted Moving Averages." *Management Science*, **6**(3), 324–342. doi:10.1287/mnsc.6.3.324.

Zhang J, Bryant J, Nissen K (2019). "Bayesian Small Area Demography." *Survey Methodology*, **45**(1).

Affiliation:

Tom Elliott
School of Health
Victoria University of Wellington
Wellington, New Zealand
and
Department of Statistics (Honorary)
University of Auckland
Auckland, New Zealand
E-mail: tom.elliott@auckland.ac.nz
URL: https://people.wgtn.ac.nz/tom.elliott

Journal of Statistical Software 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