iNZight: a GUI for Learning Statistics

Tom Elliott and Chris Wild and Daniel Barnett and Andrew Sporle

November 11, 2020

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 (**?**), 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 (**?**)
  - Jamovi (**?**)
  - ???
- Other tools targeting students:
  - NZGrapher (**?**) 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 (**?**) and 'gWidgets' (now superceeded by 'gWidgets2', (**?**)) for making graphs which react to the type of data (i.e., the user doesn't have to choose the graph type)
- Uses GTK (**?**) to produce graphical interface, accessed via the 'RGtk2' R package (**?**)
- 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 graphical user interfaces (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 (**?**).

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 (**?**) which prove a simple, platform (and indeed toolkit) independent application programming interface (API) for writing GUIs from R. The 'RGtk2' package (**?**) 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 (**?**) provides a framework-independent inteface between R and several other R packages responsible to creating GUIs, namely 'gWidgets2RGtk2' for communicating with 'RGtk2' (**?**). 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 indepen-

dent 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 (**?**) 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' (**?**), 'readxl' @citep:readxl, or 'foreign' (**?**)) 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 **??**. 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* (**?**). Further, most of the functions return the 'tidyverse' (**?**) code used behind-the-scences, providing a further stepping stone for users to learn these more complicated code workflows.

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.

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

- Several main groups of functionality:
  - data manipulation (include specifying survey structure)
  - variable manipulation
  - plot control
  - code history

4.1 Data wrangling

- 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.
- no complex "data tidying" methods - i.e., iNZight expects data to be in rectangular form (but you can move between long/wide formats)

4.2 Graphics and simple data analyses

- all powered by 'iNZightPlots' functions
  - `inzplot()` for graphics
  - `inzsummary()` for simple summary information
  - `inzinference()` for inferencial information and hypothesis testing
- uses the variable type(s) to choose the graph type. Figure x shows the available graph types
- subsetting by 1 or 2 variables is possible/easy/emphasized
  - includes a slider to cycle or 'step' between levels (can add 'motion' to graphs; (**?**))
- plot features: specific to plot type, users only see what's relevant

- some features (such as colour by) might apply to different plot types (scatter and dot plot, for example) and so selection is retained when switching between these plot types
- inferencial markup of plots
  - normal theory error bars/curves
  - or bootstrap alternative
  - "Comparison intervals" are a way of visually assessing differences - only shown on differences *that should be compared* - bar plot example
  - an approximate tukey thing? details needed here ('iNZightMR' package)
- summary information
  - a simple numeric summary *of the current plot* (philosophy: look first)
- inference information
  - again, default inferences *of the current plot*
  - both normal and bootstrap options
  - gives a list of relevant hypothesis tests (cf. other software which requires you to decide on hypothesis test first)

4.3 Saving code history

- many functions (the wrappers) generate code and attach it to their result

```
1 data.filtered <- iNZightTools::filterData(iris, "Species", "
    Setosa")
2 cat(code(data.filtered))
3 # iris %>% dplyr::filter(Species == "Setosa")
4
```

- the iNZight GUI stores the attached code after each action and appends it to the R history 'script'
- provides a record of what was done
- helps students interested in continuing data science/statistics to get familiar with code before writing it themselves (can copy+paste and modify, for example)

4.4 Analysing surveys with iNZight

- survey data:
  - iNZight gets told once, and everything else 'just works'
  - plots use survey methods and alternative plot types (histogram vs dotplot; hex bin plot vs scatter plot)
  - summary/inference all use survey methods to obtain summaries/inferences/hypothesis tests
- supports strata/cluster based survey designs, replicate weights, and post-stratification

4.5 Other modules

- A suite of other (fixed) modules for analying/exploring special types of data
  - time series
  - multiple response
  - maps
- or for doing specific tasks

  - model fitting

- each is its own "reference class" object connecting to one (or more) functions in another package
  - time series -> 'iNZightTS'
  - multiple response -> 'iNZightMR'
  - maps -> 'iNZightMaps'
  - model fitting -> 'iNZightRegression'
- these are mostly wrappers to other functions, or modified versions -> made to work as 'standalone' packages (i.e., without iNZight)
- not fully implemented, but making progress towards modules also code-writing
- also a newer add-on system allowing anyone to write new modules for iNZight (added manually or through our github-hosted repository)
- easier/faster to maintain/update
- useful for e.g., teachers of a specific course

5 Distributing and running iNZight

- effectively just a collection of R packages (our own, plus many dependencies)
- some on CRAN, others are self-hosted (so windows binaries *are* available)
- most users are new to statistics/data science, most don't know much/anything about R/computing/programming -> we ship an *all-in-one* installer for Windows

  - single directory containing R, iNZight library
  - plus shortcuts to launch R and auto-start iNZight (using `.Rprofile`)

- fully automated build:
  - single repository and github actions
  - builds source/binary files and uploads to our repository
  - builds installer and uploads to website (uses NSIS)

- CRAN release is manual (obviously)

5.1 A note for macOS and Linux users

- 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

- install the packages
- run

```
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)
- ...

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.